BGP Introduction

Introduction to BGP (Border Gateway Protocol)

Border Gateway Protocol (BGP) is the cornerstone of routing on the global internet and plays a critical role in how large networks connect and exchange routing information.

What is BGP?

- **The Internet's EGP:** BGP is the defacto **Exterior Gateway Protocol (EGP)** used to exchange Network Layer Reachability Information (NLRI) – essentially IP prefixes or routes – **between different Autonomous Systems (ASes)**. An AS is a collection of IP networks and routers under the control of one entity (like an ISP or a large enterprise).

- **Path Vector Protocol:** BGP is classified as a **Path Vector Protocol**. This is primarily because of its **AS_PATH attribute**, which lists the sequence of Autonomous Systems a route has traversed to reach its destination. This path information is crucial for loop prevention and policy application.

- **Multiprotocol BGP (MP-BGP):** Modern BGP is technically MP-BGP. This signifies its design flexibility to carry reachability information for multiple network layer protocols, not just IPv4. Common examples include:
    - IPv6 Prefixes (IPv6 Unicast)
    - MPLS VPN routes (VPNv4/VPNv6)
    - Ethernet VPN (EVPN) routes

- **The "Protocol of the Internet":** BGP is what enables the global internet by connecting different ASes. These ASes are not typically connected in a full mesh; rather, a hierarchical and policy-driven structure exists (e.g., Tier 1, Tier 2, Tier 3 ISPs).

- **Scalability:** BGP is designed to be highly scalable, capable of carrying hundreds of thousands of routes and interconnecting tens of thousands of ASes worldwide.
    - For context, the global IPv4 BGP routing table grew from around 814,000 prefixes in January 2020 to well over **900,000 (approaching 1 million)** prefixes in recent years,

and the IPv6 table is also substantial and growing.

- **Policy Implementation Tool:** BGP is more than just a routing protocol; it's a powerful **policy implementation tool**. ISPs and large organizations use BGP's rich set of path attributes (like AS_PATH, LOCAL_PREF, MED, Communities) to influence how traffic enters and exits their networks, based on peering agreements, cost, latency, and other business considerations.

Where and Who Uses BGP?

- **Between Network Carriers/ISPs:** This is the primary use case – connecting large service providers, Content Delivery Networks (CDNs), major cloud providers, and large internet exchanges to form the internet backbone.

- **Enterprises and Service Providers:** Large scale Enterprises use BGP when they are **multi-homed** (connected to two or more ISPs for redundancy and optimized path selection) or when they need to announce their own public IP address space to the internet.

BGP vs. Interior Gateway Protocols (IGPs)

BGP and IGPs (like OSPF, IS-IS, EIGRP) serve different purposes and have distinct characteristics:

Feature	BGP (Border Gateway Protocol)	IGPs (e.g., OSPF, IS-IS, EIGRP)
Primary Use	Routing **between** Autonomous Systems (ASes)	Routing **within** a single Autonomous System (AS)
Scalability	Very high; handles internet-scale routing tables (many prefixes).	Limited; designed for smaller, well-defined network domains.
Convergence	Generally slower than IGPs.	Generally faster; optimized for quick reconvergence.
Path Selection	Complex algorithm based on many path attributes (policy-driven). By default, selects one best path.	Typically based on a single metric (cost, hop count, composite metric). Easily supports Equal Cost Multi-Path (ECMP).
Neighbor Term	**Peers** or BGP Speakers.	Neighbors.
Peering	**Statically configured** between peers.	Usually **dynamically discovered** on enabled interfaces.

Peer Connectivity	**iBGP** peers (within the same AS) don't need to be directly connected. **eBGP** peers (between different ASes) are **assumed to be directly connected by default** (TTL=1), though this behavior can be modified (`ebgp-multihop`).	IGP neighbors are typically directly connected on the same subnet (exceptions like OSPF virtual links exist).
Transport	Uses **TCP port 179** (reliable, unicast communication).	OSPF/EIGRP use their Headers (89/88). IS-IS runs directly over Layer 2. RIP uses UDP 520/521. IGPs often use multicast for Hellos/discovery on broadcast segments.
Next-Hop	Advertises prefixes and a next-hop. BGP next-hop handling has specific rules (more on this later).	Advertise prefixes and a next-hop.

BGP AS Number

What is an AS Number (Autonomous System)?

- **Definition:** An AS is a network or a group of networks under a **single technical administration** that uses a common, clearly defined routing policy. From the perspective of the internet, an AS appears as a single entity.

- **Example:** A large Service Provider (ISP) operates a vast network connecting many customer organizations. When viewed from the outside (by other ISPs or networks), all these interconnected networks under the ISP's control are typically represented by a single AS Number.

- **BGP's View:** BGP treats each AS as a **single "hop"** in the AS_PATH attribute. Even if traffic must traverse multiple routers within an AS to exit towards the next AS, BGP's AS_PATH will only increment by one for that AS.

Where Can AS Numbers Be Obtained?

Public AS Numbers are allocated hierarchically:

1. **IANA (Internet Assigned Numbers Authority):** Oversees the global allocation of AS Numbers to Regional Internet Registries.

2. **RIRs (Regional Internet Registries):** These are organizations like ARIN (North America), RIPE NCC (Europe, Middle East, Central Asia), APNIC (Asia Pacific), LACNIC (Latin America, Caribbean), and AFRINIC (Africa). RIRs allocate AS Numbers to Local Internet Registries (LIRs, often ISPs) and, in some cases, directly to end-user organizations.

3. **Service Providers (ISPs):** Enterprises often obtain an AS Number as part of their service agreement with an ISP, or the ISP might use private ASNs for specific customer BGP configurations.

Who typically Needs a Public AS Number?

Organizations usually require their own public AS Number if they:

- Are **multi-homed** (connected to two or more different ISPs/ASes) and want to manage their own routing policies and path selection.

- Wish to have **Service Provider Independent (PI) addressing**, meaning their IP address block is assigned directly to them by an RIR, allowing them to change ISPs without having to renumber their network.

- Are **large network carriers, telecom operators, or significant internet content providers.**

AS Number Types and Ranges

AS Numbers were originally 16-bit, 32-bit AS Numbers were introduced later.

1. Original 16-bit AS Numbers (0 to 65535):

- **Public AS Numbers:** `1` through `64495`

- **Reserved for Documentation and Sample Code (RFC 5398):** `64496` through `64511` (16 ASNs)

- **Private Use AS Numbers (RFC 6996):** `64512` through `65534` (1023 ASNs)

- **Reserved:**
  - `0`: Reserved and should not be used.
  - `65535`: Originally reserved, now specifically designated as `AS_TRANS` (RFC 7300) for facilitating the transition between 16-bit and 32-bit AS-capable BGP speakers.

2. 32-bit AS Numbers (0 to 4,294,967,295):

The 32-bit space includes the original 16-bit range. BGP speakers supporting 32-bit ASNs can use numbers from the entire range.

- **Numbers from the 16-bit space (`0` - `65535`)** retain their original assignments and types (public, private, reserved as above).

- **Specifically 32-bit Ranges:**

    - **Reserved for Documentation and Sample Code (RFC 5398):** `65536` through `65551` (16 ASNs)

    - **Public AS Numbers (largely):** `65552` up to `4199999999` (This is the major contiguous block before the large 32-bit private range. Other smaller public blocks exist beyond this).

    - **Private Use AS Numbers (RFC 6996):** `4200000000` through `4294967294` (94,967,295 ASNs)

    - **Reserved:** `4294967295` (the last 32-bit ASN)

Support for 32-bit ASNs:

Not all routers, especially older ones, support 32-bit AS Numbers. BGP uses a capability negotiation in the OPEN message to determine if a peer supports 32-bit (4-octet) ASNs. If a router doesn't support them, special handling (like using AS_TRANS for ASNs that can't be represented in 16 bits) is required.

AS Number Notation

There are two main ways to represent AS Numbers:

1. **AS PLAIN (Plain Notation):** The AS Number is represented as a simple integer (e.g., `64512`, `65536`, `13335`). This is the most common and universally supported notation today.

2. **AS DOT (Dot Notation):** Originally introduced to help represent the new 32-bit ASNs in a way that older systems or displays might handle, or for human readability (e.g., `X.Y` where `X * 65536 + Y = AS_PLAIN_VALUE`). For example, ASN `65537` can be written as `1.1`.

- While `asplain` is now preferred for configuration, `asdot` notation might still be encountered in some logs, older documentation, or specific vendor displays. It's not "obsolete" but is less common for primary configuration than `asplain`.

Public AS vs. Private AS Usage

The distinction is similar to public vs. private (RFC 1918) IPv4 addresses:

- **Private AS Numbers:** Can be used freely within a lab environment or for internal BGP configurations within an organization where routes are not intended to be advertised to the global internet (e.g., between an enterprise and its ISP if the ISP strips/changes the private ASN before advertising to other peers).

- **Public AS Numbers:** Required if an organization intends to exchange BGP routing information on the public internet. ISPs will filter and drop BGP updates containing private AS Numbers in the AS_PATH when receiving them from public peers.

If you plan to connect to the internet using BGP and announce your own routes, you must obtain a public AS Number and public IP address space from the appropriate RIR or through your ISP.

BGP Connectivity Scenarios

How Enterprises Use BGP with Service Providers

When an enterprise network connects to Service Providers, the need for BGP depends on the complexity and redundancy of these connections.

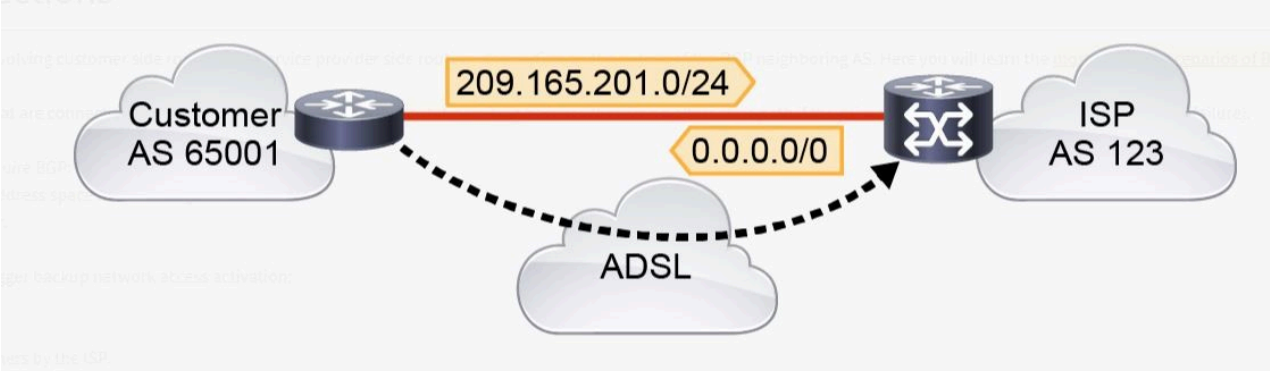
- **Simple Scenario (No BGP Needed):** If an enterprise has only a **single connection** from its edge router to a single Service Provider, BGP is often unnecessary. A **static default route** on the enterprise router pointing to the ISP's router is usually sufficient and simpler to manage.

- **More Complex Scenarios (BGP Becomes Valuable):** When an enterprise has multiple connections, perhaps to different Service Providers, BGP becomes essential. It allows the enterprise to:
  - Advertise its own public IP address space (Provider Independent - PI, or Provider Assigned - PA if conditions allow).
  - Implement routing policies to influence how traffic enters (inbound) and exits (outbound) its network.
  - Achieve redundancy and automatic failover.
  - Potentially load-balance traffic across multiple links or ISPs.

Common Enterprise BGP Connectivity Scenarios

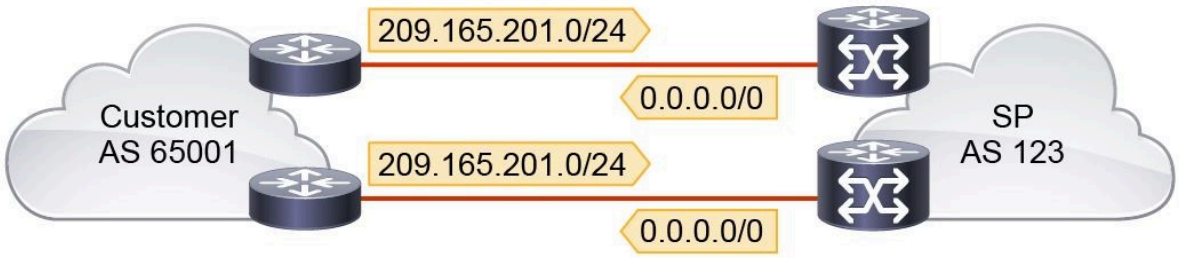
Here are typical ways enterprises connect to ISPs using BGP:

1. **Single-Homed (1 Link, 1 ISP)**
   - **Description:** The enterprise has a single link to a single ISP.
   - **BGP Usage:** Typically, BGP is **not recommended** here due to added complexity without significant benefit. A static default route from the enterprise to the ISP is the preferred, simpler solution. The ISP might advertise the enterprise's prefixes to the internet.



2. **Dual-Homed (2+ Links, 1 ISP)**

- **Description:** The enterprise has two or more links connecting to the **same** ISP. These links can terminate on a single enterprise router or on two separate enterprise routers for higher availability.
- **BGP Usage:** BGP **can be used** here. It allows for:
  - **Redundancy:** If one link fails, BGP can automatically reroute traffic over the remaining link(s).
  - **Path Selection/Load Balancing:** The enterprise can influence outbound path selection or potentially load-balance traffic across the links (though BGP's primary load balancing is per-prefix, not per-flow by default).
  - The ISP might provide default routes or partial/full routes.



3. **Single-Multihomed (1 Link per ISP, 2+ ISPs)**
   - **Description:** The enterprise has single links to two or more **different** ISPs.
   - **BGP Usage:** BGP is **highly recommended**.
     - **Redundancy:** If one ISP connection fails, traffic can be routed via the other ISP(s).
     - **Policy Control:** The enterprise can advertise its prefixes to both ISPs and implement policies to prefer one ISP for certain traffic or as a primary/backup.
     - The enterprise usually receives at least a default route from each ISP, and often partial or full internet routes if desired.

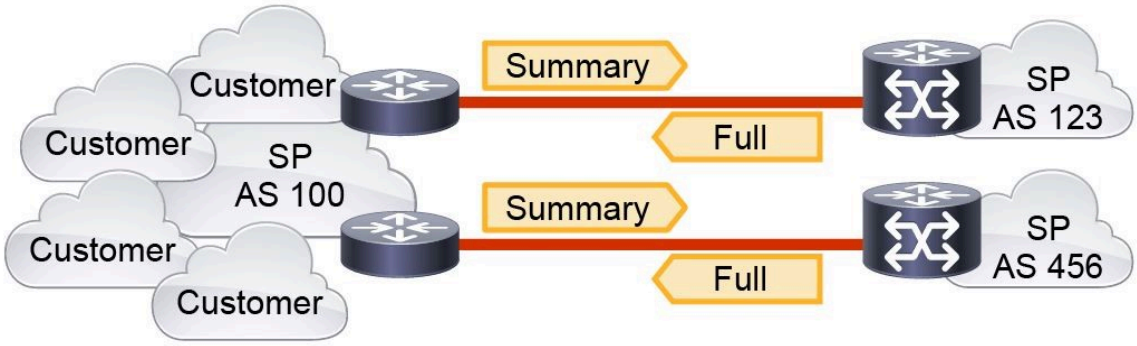
4. **Dual-Multihomed (2+ Links per ISP, 2+ ISPs)**
   - **Description:** This is the most resilient and flexible setup. The enterprise has multiple links to multiple different ISPs (e.g., two links to ISP A and two links to ISP B).

- **BGP Usage:** BGP is **essential** to leverage the full benefits:
  - **Maximum Redundancy:** Protects against individual link failures and complete ISP outages.
  - **Sophisticated Policy Control:** Allows for granular control over inbound and outbound traffic paths, load balancing, and provider preference.
  - Design options are varied (e.g., all links to one enterprise router, dedicated routers per ISP connection, etc.).

Some ISPs Scenarios

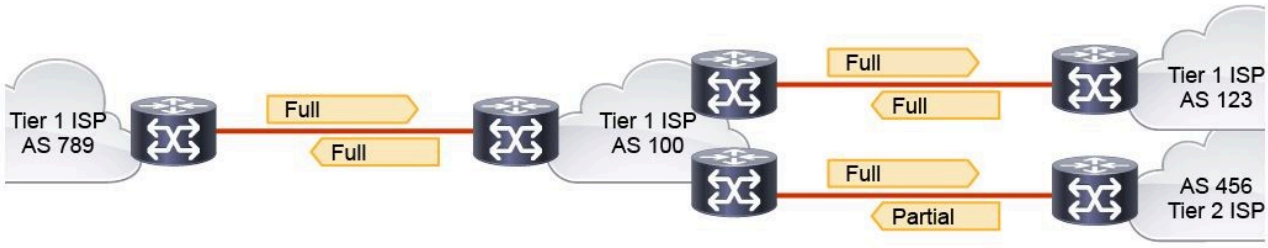
ISPs form the backbone of the internet, and BGP is the protocol that glues them together.

1. **Peering with Upstream Providers and Other ISPs (ISP-to-ISP):**
   - ISPs connect to each other in a tiered fashion (Tier 1, Tier 2, Tier 3).
     - **Tier 1 ISPs** have access to the entire internet routing table via settlement-free peering with other Tier 1 providers. They form the core internet backbone.
     - **Tier 2 ISPs** typically peer with some networks (often settlement-free for regional traffic) and purchase "transit" (access to the full internet routing table) from Tier 1 or larger Tier 2 providers. They often provide transit to Tier 3 ISPs and enterprise customers.
     - **Tier 3 ISPs** primarily purchase transit from Tier 2 or Tier 1 providers and serve end-users and smaller businesses.
   - **Type of Peering:**
     - **Upstream Provider (Transit):** An ISP receives the full internet routing table from its upstream provider. The upstream provider, in turn, learns the customer ISP's routes (and its customers' routes) and advertises them globally. This is usually a paid service.
     - **Peering (Settlement-Free or Paid):** ISPs of similar size or strategic importance might agree to exchange only their respective customer routes with each other, often without payment (settlement-free) or under specific financial terms. This is common at Internet Exchange Points (IXPs).



2. **Transit ISPs:**

- This describes the role of an ISP (often Tier 1 or large Tier 2) that provides transit services. They exchange full internet routing tables with their peers and downstream customers (other ISPs).
- Tier 1 ISPs send the full routing table to lower-tier ISPs they provide transit to.
- Lower-tier ISPs advertise their customer prefixes (and their own) up to their transit provider(s).



External BGP vs. Internal BGP

BGP operates in two primary modes depending on whether peering is established between different Autonomous Systems (ASes) or within the same AS.

eBGP (External BGP) - Peering Between ASes

eBGP is used when BGP routers in **different Autonomous Systems** form a peer relationship to exchange routing information.

1. **Inter-AS Routing:** Its fundamental role is to connect distinct ASes, allowing them to advertise and learn network reachability information from each other.

2. **Peer Identification:** A BGP router determines if a peer is "external" by checking the `remote-as <ASN>` configuration for that neighbor. If the remote ASN is different from the local router's ASN, the session is eBGP.

3. **Direct Connectivity (Default):** By default, eBGP peers are assumed to be **directly connected** (i.e., on the same subnet). Consequently, the Time-To-Live (TTL) value in the IP header of eBGP packets is set to **1**. This behavior can be overridden (e.g., using `ebgp-multihop`) if peers are not directly connected, though this is less common for typical inter-AS links.

4. **AS_PATH for Loop Prevention:** The primary loop prevention mechanism in eBGP is the **AS_PATH attribute**. When a route update is received from an eBGP peer, the receiving router checks if its own AS Number is already present in the AS_PATH list. If it is, the update is dropped to prevent a routing loop. This behavior can also be modified (e.g., with `allowas-in`), but that's for specific scenarios only.

5. **Next-Hop Modification:** When an eBGP router advertises a route to an eBGP peer, it **changes the NEXT_HOP attribute to its own IP address** (typically the IP address of the interface used for the peering session or a specified update source).

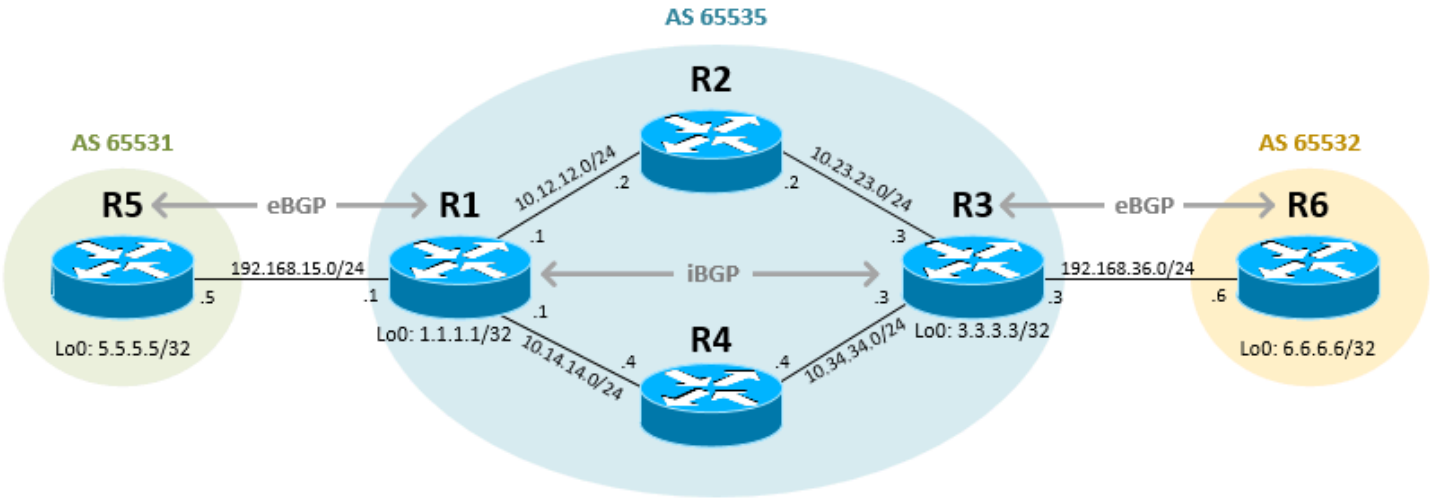
---

iBGP (Internal BGP) - Peering Within an AS

iBGP is used when BGP routers **within the same Autonomous System** form peer relationships. Its main job is to ensure that all BGP routers inside your AS have a consistent understanding of routes learned from *outside* your AS.

1. The Scenario: Learning External Routes

Let's look at AS 65535, which contains routers R1, R2, R3, and R4.



- Router **R1** is an edge router. It forms an **eBGP** peering with **R5** (in AS 65531).
- Through this eBGP session, R1 learns external routes from AS 65531 – for example, the route to R5's loopback, `5.5.5.5/32`.

The Core Problem: Now that R1 knows how to reach `5.5.5.5/32`, how do the other routers *inside* AS 65535 (R2, R3, and R4) learn about this route? if:

- Internal routers like R2 or R4 need to send traffic to `5.5.5.5/32`.
- Another edge router, like **R3** (which peers via eBGP with **R6** in AS 65532), needs to know about `5.5.5.5/32` so it can potentially advertise it further to AS 65532.

2. Why iBGP? Propagating External Routes Internally

How does R1 share the knowledge of `5.5.5.5/32` with R2, R3, and R4?

- **Option A: Redistribute BGP into your IGP (e.g., OSPF, IS-IS) within AS 65535?**
  - This is **strongly discouraged and generally not valid Option** for full internet routing tables. IGPs are designed for routing *within* an AS and are not built to handle the massive scale (hundreds of thousands to millions of prefixes) of BGP routes. Attempting this would likely crash your IGP.

- **Option B: Use Internal BGP (iBGP).**
  - **This is the correct solution.** R1 will form **iBGP peerings** with R2, R3, and R4 (and they will peer among themselves as needed).
  - R1 then advertises the route to `5.5.5.5/32` (which it learned via eBGP from R5) to its iBGP peers R2, R3, and R4.
  - Thus, iBGP is the protocol responsible for carrying these *external* reachability details *across* your own AS (AS 65535).

3. So Why need an IGP then?

So, iBGP carries the external route information (like `5.5.5.5/32`) between R1, R2, R3, and R4. But there's a catch:

- **iBGP Peering:** BGP, including iBGP, runs over **TCP port 179**. For R1 to establish a TCP session with R3 for their iBGP peering, R1 must be able to reach R3's IP address that is used for peering (typically a loopback address, like `1.1.1.1/32` for R1 and `3.3.3.3/32` for R3).

- **iBGP Peers Often Not Directly Connected:** In the diagram, R1 is not directly connected to R3. The path might go R1 -> R2 -> R3, or R1 -> R4 -> R3.

This is where your **Interior Gateway Protocol (IGP)**, such as OSPF or IS-IS, running within AS 65535, is essential:

- **Underlying IP Reachability:** The IGP ensures that all routers within AS 65535 (R1, R2, R3, R4) can reach each other's internal IP addresses, including the loopback addresses they use for iBGP peering. This allows the iBGP TCP sessions to be established.

- **Next-Hop Resolution:** When R1 advertises the route to `5.5.5.5/32` to R2, R3, and R4, the BGP next-hop for this route will (by default) be the IP address of the eBGP peer R5. Your IGP must ensure that all routers within AS 65535 know how to reach this next-hop (or R1 must be configured with `next-hop-self` to change the next-hop to itself when advertising to iBGP peers).

In summary: iBGP distributes the *external* BGP routes within your AS, while the IGP provides the *internal* transport network that allows iBGP peers to communicate and resolve BGP next-hops.

4. iBGP Loop Prevention: The Split-Horizon Rule

eBGP uses the AS_PATH to prevent loops between ASes. iBGP needs its own mechanism because the **AS_PATH is NOT modified when routes are passed between iBGP peers** within the same AS.

- **The Rule:** A route learned from an iBGP peer **cannot be advertised to another iBGP peer.**
    - In AS 65535: If R1 learns the route to `5.5.5.5/32` from R5 (eBGP) and advertises it to its iBGP peer R2. R2 **will not** advertise this route to its *other* iBGP peer, R3 (or R4).

- **Why?** Without this rule, since AS_PATH isn't changing, the route could be passed around in a loop among iBGP speakers indefinitely.

- **Consequence:** This means that for R3 to learn the route to `5.5.5.5/32`, it must either peer directly with R1 (the source of the iBGP information for this route within the AS) or receive it via a mechanism that bypasses standard iBGP split horizon.

- **Solutions:**

1. **Full Mesh:** Every iBGP router peers with every other iBGP router in AS 65535. (R1 peers with R2, R3, R4; R2 peers with R1, R3, R4; etc.). This ensures all routers get all iBGP updates. Manageable for a few routers like in the diagram, but scales poorly (N*(N-1)/2 sessions).

2. **Route Reflectors (RRs):** A more scalable solution. One or more routers (e.g., R1 or a dedicated router) can be configured as an RR. The RR *can* reflect routes learned from one iBGP peer (a "client") to other iBGP peers ("clients").

3. **BGP Confederations:** Less common, involves dividing the AS into sub-ASes.

5. Other Key iBGP Characteristics

- **Next-Hop Unchanged (Default):** When R1 (in AS 65535) learns the route to `5.5.5.5/32` from R5 (AS 65531), the next-hop is R5's IP address. When R1 advertises this route to its iBGP peers (R2, R3, R4), this next-hop attribute **remains R5's IP address by default**.
    - This means R2, R3, and R4 must have a route (via your IGP) to reach R5's IP address. Alternatively, R1 can be configured with the `next-hop-self` command for its iBGP peers, causing R1 to advertise itself as the next-hop.

- **TTL (Time-To-Live):** Since iBGP peers (like R1 and R3) might be multiple internal hops away from each other, iBGP packets are sent with a high TTL (e.g., 255) to ensure they can traverse the internal network. This is different from eBGP's default TTL of 1.

BGP Peering

BGP Workflow

BGP's workflow shares a high-level similarity with Interior Gateway Protocols (IGPs), but its operational details and design philosophy are quite different, primarily because it's built for inter-domain routing with a strong emphasis on policy.

Similar Workflow to IGPs:

1. **BGP Peering:** BGP routers establish peer relationships (similar to IGP neighborships).

2. **Exchange Reachability Information:** Peers exchange Network Layer Reachability Information (NLRI), which are essentially IP prefixes (routes).

3. **Best Path Selection & Propagation:** Each BGP router runs a best path selection algorithm on the received routes, installs the best path(s) into its BGP table (and potentially its global routing table - RIB), and then advertises valid paths to its other BGP peers based on policy.

Key Differences from IGPs:

1. **Peering Establishment:**
    - **IGPs (OSPF, EIGRP, IS-IS):** Typically use multicast or broadcast Hello packets on enabled interfaces to dynamically discover neighbors and form adjacencies.
    - **BGP:** Peerships are **statically configured**. You must explicitly define each neighbor's IP address and their Autonomous System (AS) number. BGP communication is unicast over TCP.

2. **Route Advertisement:**
    - **IGPs:** Generally advertise all learned routes (or routes for interfaces configured for the IGP) automatically to their neighbors within the same routing domain.
    - **BGP:** Is more selective. By default, it doesn't advertise anything. You must explicitly tell BGP which prefixes to originate (e.g., using the `network` command or through redistribution) or it will advertise valid BGP routes it has learned from other BGP peers and selected as best.

3. **Path Selection Algorithm:**
    - **IGPs:** Use algorithms like Dijkstra (OSPF, IS-IS), DUAL (EIGRP), or Bellman-Ford (RIP), primarily focused on finding the shortest path based on a metric (cost, hop count, etc.). They are often designed for easy load balancing over equal-cost paths.
    - **BGP:** Uses a complex **Best Path Selection Algorithm** that evaluates numerous **Path Attributes** (e.g., AS_PATH, LOCAL_PREF, MED, Origin, Weight). It's heavily geared towards policy enforcement and

typically selects only a single best path to a destination by default (though features like BGP Multipath exist).

How eBGP Peering is Formed

External BGP (eBGP) is used between routers in different Autonomous Systems.
Basic Configuration:

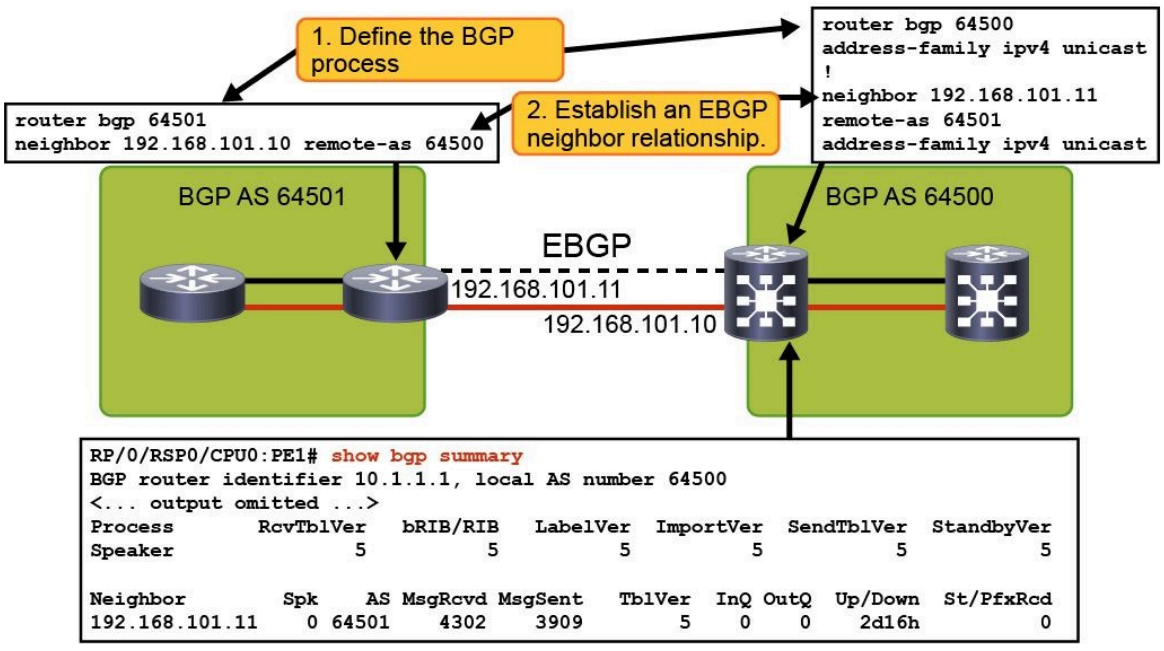
- **IOS XE:**

```
R1(config)# router bgp <LOCAL_AS_NUMBER>
R1(config-router)# neighbor <NEIGHBOR_IP_ADDRESS> remote-as <NEIGHBOR_AS_NUMBER>
```

- **IOS XR:**

```
XR1(config)# router bgp <LOCAL_AS_NUMBER>
XR1(config-bgp)# address-family ipv4 unicast  // Enable BGP for IPv4 globally
XR1(config-bgp-af)# exit
XR1(config-bgp)# neighbor <NEIGHBOR_IP_ADDRESS>
XR1(config-bgp-neighbor)# remote-as <NEIGHBOR_AS_NUMBER>
XR1(config-bgp-neighbor)# address-family ipv4 unicast // Activate IPv4 for this neighbor
XR1(config-bgp-neighbor-af)# exit
```

(Note: On IOS XR, BGP needs a Router ID to be explicitly configured or dynamically derived. If it's not available, the BGP process might not start correctly, or TCP connections might reset)



With this basic eBGP configuration, the router assumes:

1. **eBGP Session:** Since the `remote-as` is different from the local BGP AS number, this is an eBGP peering.
2. **TCP Connection:** The router will attempt to establish a TCP session to the neighbor's IP address on **port 179** and also listen for incoming connections on port 179.
3. **TTL=1:** For eBGP, packets are sent with an IP Time-To-Live (TTL) of **1**, assuming neighbors are directly connected.
4. **Source IP & Reachability:** The router will use the IP address of the outgoing interface closest to the neighbor as the source for BGP packets, unless `update-source` is configured. It expects the neighbor's IP address to be on a directly connected network.

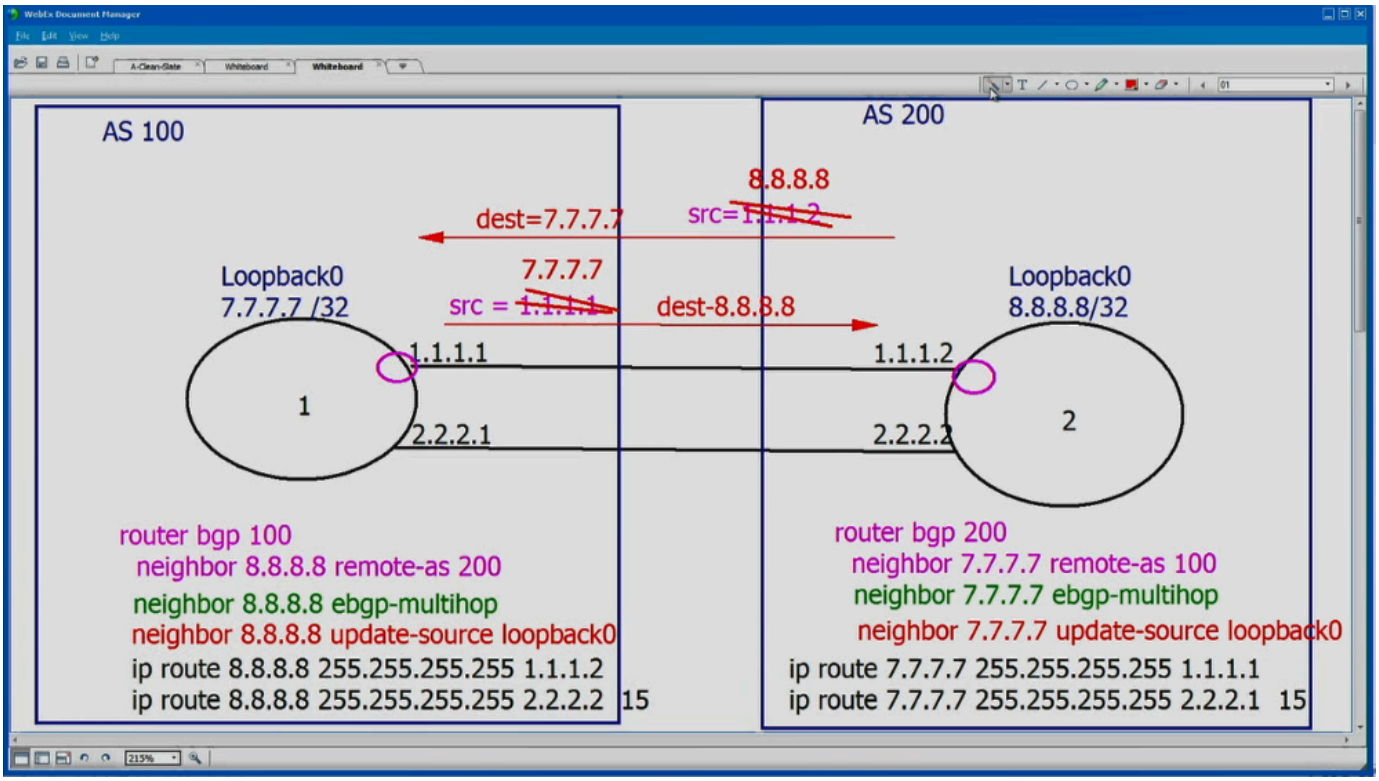
Troubleshooting eBGP Peering

Common issues preventing eBGP peering:

1. **Reachability:** No route to the neighbor's IP address, or neighbor is not on a directly connected segment (and `ebgp-multihop` is not used).
2. **TCP Port 179 Blocked:** Firewalls or Access Control Lists (ACLs) blocking TCP port 179.
3. **AS Number Mismatch:** Incorrect `remote-as` configured for the neighbor (syslog messages often indicate "peer in wrong AS").
4. **IP Address Mismatch:** Incorrect neighbor IP address configured.
5. **Duplicate BGP Router IDs:** Router IDs must be unique.
6. **Authentication Failure:** If BGP MD5 authentication is configured, passwords must match.
7. **Address Family Not Activated (IOS XR):** Forgetting to activate the relevant address family (e.g., `ipv4 unicast`) for the neighbor in IOS XR.

`ebgp-multihop` & `update-source`

These are essential when eBGP peers are not directly connected or when peering using loopback interfaces (a common best practice for stability).



1. `neighbor <NEIGHBOR_IP> ebgp-multihop [TTL_VALUE]`
    - **Purpose:** Increases the TTL of eBGP packets sent to this neighbor. This is necessary if the peer is more than one IP hop away. If not specified, `[TTL_VALUE]` defaults to 255.
    - **Usage:** Typically used when peering between loopback interfaces.

2. `neighbor <NEIGHBOR_IP> update-source <INTERFACE_NAME>` (e.g., `update-source Loopback0`)
    - **Purpose:** Specifies which local IP address (and thus interface) should be used as the source IP for BGP packets sent to this neighbor.
    - **Usage:** Crucial when peering with loopback addresses to ensure a consistent and reliable source IP.

Requirements when using these commands:

- You **must have a static route or an IGP route** (anything *except* a BGP-learned route or default route) to reach the neighbor's IP address specified in the `neighbor` command (which is now often their loopback address).

Internal BGP (iBGP) Peering

When the AS number in the `router bgp <ASN>` command is the **same** as the `remote-as <ASN>` in the `neighbor` command, the router knows this is an iBGP peer. iBGP has different rules:

1. **TTL=255:** Packets are sent with a TTL of 255 by default, as iBGP peers are often not directly connected and rely on an IGP for reachability within the AS.

2. **AS_PATH Unchanged:** The AS_PATH attribute is not modified when sending updates to iBGP peers.

3. **Split Horizon:** A route learned from one iBGP peer is NOT advertised to another iBGP peer (this is the iBGP loop prevention mechanism).

4. **Next-Hop Unchanged (Default):** The NEXT_HOP attribute learned from an eBGP peer is generally not changed when advertised to iBGP peers (can be modified with `next-hop-self`).

Key BGP Peering Configuration Commands

- **IOS XE:**

```
R1(config)# router bgp <LOCAL_ASN>
R1(config-router)# neighbor <NEIGHBOR_IP> remote-as <REMOTE_ASN>
R1(config-router)# neighbor <NEIGHBOR_IP> ebgp-multihop <TTL_VALUE_IF_NEEDED>
R1(config-router)# neighbor <NEIGHBOR_IP> update-source Loopback<NUMBER>
```

- **IOS XR:**

```
XR1(config)# router bgp <LOCAL_ASN>
XR1(config-bgp)# neighbor <NEIGHBOR_IP>
XR1(config-bgp-neighbor)# remote-as <REMOTE_ASN>
XR1(config-bgp-neighbor)# ebgp-multihop <TTL_VALUE_IF_NEEDED>
XR1(config-bgp-neighbor)# update-source Loopback<NUMBER>
XR1(config-bgp-neighbor)# address-family ipv4 unicast // Or other AFs
XR1(config-bgp-neighbor-af)# exit
```

Verifying BGP Peering State

- **IOS XE & IOS XR:**
  - `show bgp [ipv4 unicast] summary` (IOS XE often uses `show ip bgp summary`)

```
R3#
R3#
R3#show ip bgp summary
BGP router identifier 33.33.33.33, local AS number 1
BGP table version is 1, main routing table version 1

Neighbor        V           AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
11.11.11.11     4            1       7       7        1    0    0 00:03:05        0
R3#
```

  - `show bgp [ipv4 unicast] neighbors [NEIGHBOR_IP]` (IOS XE `show ip bgp neighbors`)

```
R3#show ip bgp neighbors
BGP neighbor is 11.11.11.11,  remote AS 1, internal link
  BGP version 4, remote router ID 11.11.11.11
  BGP state = Established, up for 00:02:28
  Last read 00:00:35, last write 00:00:40, hold time is 180, keepalive interval is 60 seconds
  Neighbor sessions:
    1 active, is not multisession capable (disabled)
  Neighbor capabilities:
    Route refresh: advertised and received(new)
    Four-octets ASN Capability: advertised and received
    Address family IPv4 Unicast: advertised and received
    Enhanced Refresh Capability: advertised and received
    Multisession Capability:
    Stateful switchover support enabled: NO for session 1
  Message statistics:
    InQ depth is 0
    OutQ depth is 0
```

Checking the Underlying TCP Session

Since BGP runs over TCP port 179, you can verify TCP session establishment:

- **IOS XE & IOS XR:**

```
R1# show tcp brief
```

- **Clearing a TCP Session (forces TCP teardown and re-establishment):**
  - **IOS XE:** `clear tcp tcb <value>`
  - **IOS XR:** `clear tcp pcb <value>`

(It's generally better to clear the BGP session itself rather than trying to clear the underlying TCP TCB/PCB directly for BGP troubleshooting, as clearing BGP handles the protocol state reset properly.)

BGP Peer Flapping

The Scenario and Problem Diagnosis

You have eBGP peers using their loopback interfaces (which are reachable via an IGP like OSPF) as the source for their BGP sessions. The issue arises when these same loopback IP prefixes are also advertised *into BGP* and exchanged between the eBGP peers.

1. **Initial State:** OSPF (or another IGP) provides reachability to the loopback IP addresses of all routers. These OSPF routes have an Administrative Distance (AD) of, for example, 110. This allows the BGP TCP sessions between the loopbacks to establish.

2. **BGP Peering Up:** The eBGP sessions come up.

3. **If Loopbacks Advertised via eBGP:** The routers advertise their own loopback prefixes into BGP. When Router A receives Router B's loopback prefix via eBGP, this route has an AD of 20.

4. **RIB Decision:** Since AD 20 (eBGP) is better (lower) than AD 110 (OSPF), Router A replaces its OSPF route to Router B's loopback with the eBGP-learned route. The same happens on Router B for Router A's loopback.

5. **The Recursive Failure:** Now, Router A believes the path to Router B's loopback (which is the IP address it needs to maintain its BGP session with Router B) is *via BGP itself*. BGP requires its underlying transport (the route to the peer's IP address) to be stable and provided by a non-BGP source (IGP or static).

6. **Session Flap:** BGP keepalives will fail because the route to the peer's `update-source` (the loopback) is now dependent on the BGP session itself. The hold-down timer expires, the BGP session drops, and then it attempts to re-establish using the IGP routes again, restarting the cycle.

How to Solve This?

1. Best Solution: Do Not Advertise Peering Loopbacks into eBGP to Each Other

- The simplest and cleanest solution is to **filter** or simply **not advertise** the specific /32 loopback prefixes used for eBGP peering into the eBGP session they support. These loopbacks are infrastructure addresses for establishing the BGP session; they typically don't need to be advertised *over that specific eBGP session* to that specific peer. Let the IGP handle their reachability internally and between the eBGP edge routers.

2. Less Common: Change Administrative Distances

- You could manually change the AD of the OSPF routes for these specific loopback prefixes to be lower than 20 (e.g., AD 15). This would make the OSPF route always preferred over the eBGP learned route for those loopbacks. While possible, it's generally less clean than solution #1.

3. Using the `network x.x.x.x mask z.z.z.z backdoor` Command (IOS XE)

- This command is used when a router is originating a prefix into BGP using the `network` command, and that same prefix is *also known via an IGP* on that router.

- The `backdoor` keyword makes the BGP path for that prefix less preferable *in the local router's RIB* compared to an IGP route to the same prefix.

  IOS XE:

  ```
  R1(config)# router bgp <YOUR_ASN>
  R1(config-router)# network <LOOPBACK_IP> mask <LOOPBACK_MASK> backdoor
  ```

  IOS XR

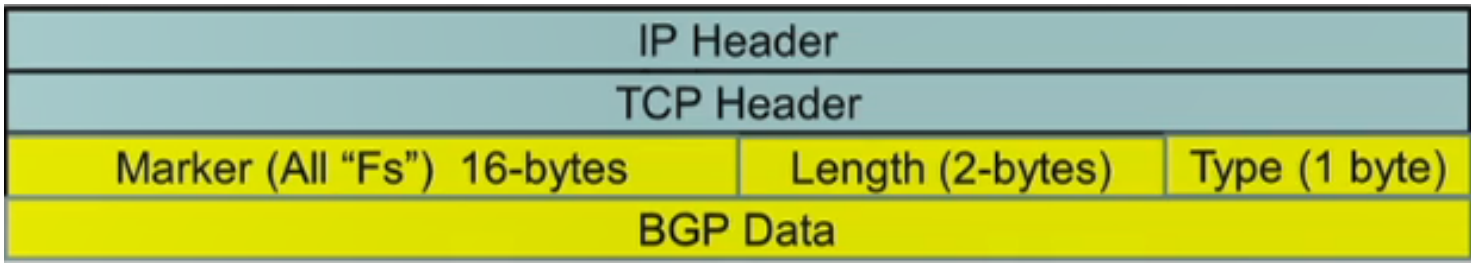
  IOS XR does not have a direct one-to-one network ... backdoor command, but the same behavior can be achieved using route policy

BGP Packet types

BGP communicates using five main message types, all encapsulated within IP/TCP headers. Each BGP message starts with a common 19-byte header.

BGP Common Message Header

All BGP messages share this header structure:



1. **Marker (16 bytes):**
   - This field is typically all ones (`0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF`).
   - Its primary purpose is to help receiving routers detect loss of synchronization and delineate BGP messages.

2. **Length (2 bytes):**
   - Indicates the **total length of the BGP message in octets, including the 19-byte common header**.
   - The minimum length is 19 bytes (for a KEEPALIVE message), and the maximum is 4096 bytes.

3. **Type (1 byte):**
   - Specifies the BGP message type. The defined types are:
     - **1: OPEN**
     - **2: UPDATE**
     - **3: NOTIFICATION**
     - **4: KEEPALIVE**
     - **5: ROUTE-REFRESH** (defined in RFC 2918)

---

1. OPEN Message (Type 1)

This is the first message sent after the TCP connection is established. It's used to negotiate the parameters and capabilities for the BGP peering session.

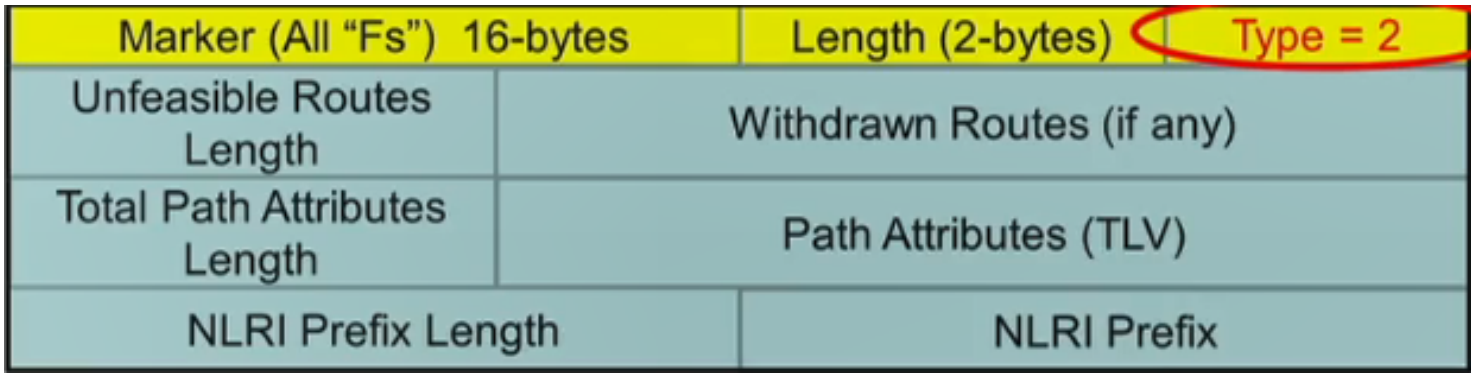
- **Key Fields:**
  - `Version`: BGP version number (current is 4).
  - `My Autonomous System`: The sender's AS number.
  - `Hold Time`: The maximum time (in seconds) the sender proposes to wait without receiving a KEEPALIVE or UPDATE message before considering the peer down. The peers will negotiate to use the lower of their proposed timers.
  - `BGP Identifier`: The sender's BGP Router ID (a 32-bit ID, usually an IP address).
  - `Optional Parameters Length`: Length of the Optional Parameters field.
  - `BGP Capabilities`: Used to advertise capabilities like support for Multiprotocol Extensions (MP-BGP for IPv6, VPNs, etc.), 32-bit (4-octet) AS numbers, Route Refresh, Graceful Restart, etc.
- **Peering Establishment:** For a BGP peering to be established, critical parameters in the OPEN messages from both peers must be compatible (e.g., BGP version, and the peer's AS number must match the `remote-as` configured locally). Capabilities are also exchanged and acknowledged here.

---

2. UPDATE Message (Type 2)

This message is the workhorse of BGP, used to exchange routing information. An UPDATE message can:

- Advertise new routes.
- Withdraw previously advertised routes.



- **Key Components:**
  - `Withdrawn Routes Length` (2 bytes): Length of the Withdrawn Routes field.
  - `Withdrawn Routes` (variable): A list of IP address prefixes that are no longer reachable and should be removed from BGP routing tables. This is BGP's way of "poisoning" or retracting routes.
  - `Total Path Attributes Length` (2 bytes): Length of the Path Attributes field.
  - `Path Attributes` (variable): A set of attributes (like AS_PATH, NEXT_HOP, ORIGIN, MED, LOCAL_PREF, etc.) that apply to all prefixes listed in the NLRI field of *this specific* UPDATE message. These are structured similarly to TLVs.
  - `Network Layer Reachability Information (NLRI)` (variable): A list of IP address prefixes being advertised as reachable, sharing the accompanying Path Attributes.

---

3. NOTIFICATION Message (Type 3)

This message is sent when a BGP error condition is detected. After sending a NOTIFICATION message, the BGP session (and the underlying TCP connection) is **closed** using TCP Reset flag.

- **Key Fields:**
  - `Error Code` (1 byte): Indicates the general category of the error (e.g., Message Header Error, OPEN Message Error, UPDATE Message Error, Hold Timer Expired, Finite State Machine Error,

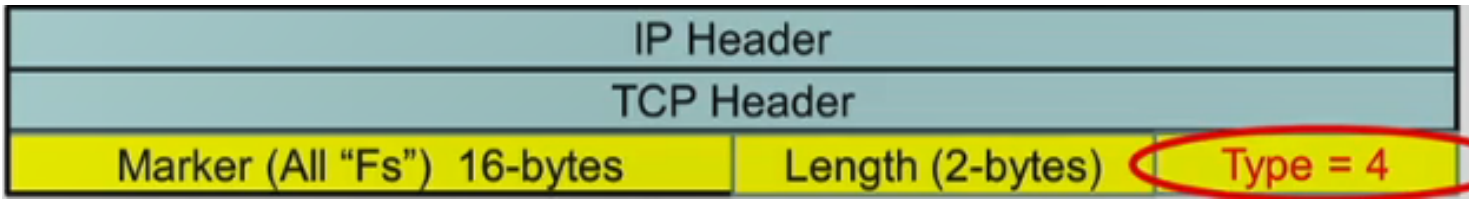
Cease).

- `Error Subcode` (1 byte): Provides more specific information about the nature of the error.
- `Data` (variable): Contains optional data related to the error, often including the part of the message that caused the error.

---

4. KEEPALIVE Message (Type 4)

Once a BGP session is established, peers exchange KEEPALIVE messages periodically to ensure the session remains active and the peer is reachable.



- **Purpose:** Similar to Hello messages in IGPs.
- **Content:** Consists only of the 19-byte BGP common header. It contains no additional data.
- **Timers:**
  - Default KEEPALIVE interval: **60 seconds**.
  - Default Hold Time: **180 seconds** (typically 3 times the KEEPALIVE interval). If a router doesn't receive a KEEPALIVE or UPDATE from its peer within the Hold Time, it declares the peer down and closes the session.

---

5. ROUTE-REFRESH Message (Type 5)

Defined in RFC 2918, this message allows a BGP speaker to request a peer to re-advertise its routes for a particular Address Family Identifier (AFI) / Subsequent Address Family Identifier (SAFI).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+-------------------------------+
|                             Marker                            |
|              (16 bytes, all 1s: 0xFFFFFFFFFFFFFFFF...)        |
+---------------+---------------+---------------+---------------+
|        Length (23)            |     Type (5)  |  AFI (high)   |
+---------------+---------------+---------------+---------------+
```

```
|  AFI (low)    |   Reserved    |     SAFI      |
+---------------+---------------+---------------+
```

- **Purpose:** Useful when routing policies change, allowing the router to receive a fresh set of routes from its peer without needing to reset the entire BGP session.

- **Capability Negotiation:** Support for Route Refresh is advertised as a capability in the OPEN message.

BGP FSM

BGP Finite State Machine (FSM)

BGP establishes and maintains peer relationships through a series of states. When you configure a BGP neighbor, the local BGP process attempts to progress through these states to reach the "Established" state, where routes can be exchanged.
Here are the key states in the BGP FSM:

1. **IDLE State**

    - **Meaning:** This is the initial state of the BGP FSM. In this state, the BGP process:
        - Refuses all incoming BGP connection attempts from the specified peer.
        - Is not actively trying to initiate a TCP connection to the peer.
        - Has no BGP resources allocated for this peer.

    - **Triggers to move out of Idle:** A "start event" occurs, such as:
        - An administrator clearing a previous BGP session.

    - **Reasons for potentially staying in Idle:**
        - The BGP process is administratively shut down for that peer.
        - No valid route exists in the routing table to reach the configured BGP neighbor's IP address (so TCP connection cannot even be attempted).
        - For eBGP, if the `ebgp-multihop` command is missing and the neighbor is not directly connected.

2. **Connect State**

    - **Meaning:** BGP has initiated a TCP connection attempt to the remote peer and is waiting for the TCP three-way handshake to complete. The `ConnectRetryTimer` is active.

- **Successful Transition:** If the TCP connection is successful, BGP sends an OPEN message to the peer and transitions to the **OpenSent** state. The ConnectRetryTimer is cleared.

- **Failed Transition:**
  - If the TCP connection attempt fails (e.g., peer unreachable, port 179 closed on peer, network issue), BGP transitions to the **Active** state.
  - If the ConnectRetryTimer expires while still in the Connect state, BGP also transitions to the **Active** state and continues to listen for an incoming connection from the peer.

- **Visibility:** This state can be very brief and might not always be visible in `show` commands but will appear in BGP FSM debugs.

3. **Active State**

- **Meaning:** BGP is actively trying to establish a TCP connection with the peer, but previous attempts have failed.

- **Actions:**
  - BGP resets the ConnectRetryTimer and transitions back to the **Connect** state to re-attempt TCP connection.
  - It also actively listens for an incoming TCP connection from the peer on port 179.

- **Reasons for getting stuck in Active:**
  - Persistent TCP connectivity issues (e.g., no route to peer, firewall/ACL blocking TCP port 179, incorrect peer IP address).
  - BGP not configured (or not running) on the remote peer.
  - Physical layer or data link layer issues.

- **Troubleshooting:** Commands like `show tcp brief` (on both XE and XR) are useful here to see if TCP sessions are even attempting to form or are stuck.

4. **OpenSent State**

- **Meaning:** The TCP connection has been successfully established, and the local BGP router has sent an OPEN message to its peer. It is now waiting to receive an OPEN message back from the peer.

- **Parameters in OPEN:** The OPEN message contains crucial parameters for negotiation, such as BGP version, local AS number, hold time, BGP identifier (router ID), and optional capabilities (like MP-BGP support, 4-octet ASN support, Route Refresh).

- **If an OPEN message is received from the peer:** Basic validation is performed (e.g., BGP version, AS number matches configured `remote-as`, BGP ID is unique). If valid, BGP sends a KEEPALIVE message and transitions to **OpenConfirm**. If invalid, BGP sends a NOTIFICATION message and goes back to **Idle**.

5. **OpenConfirm State**

- **Meaning:** The local BGP router has sent its OPEN message and has also successfully received and validated an OPEN message from its peer. The local router has sent its first KEEPALIVE message.

- **Action:** It is now waiting for a KEEPALIVE message from its peer to confirm that the peer also accepts the parameters and is ready to establish the session.

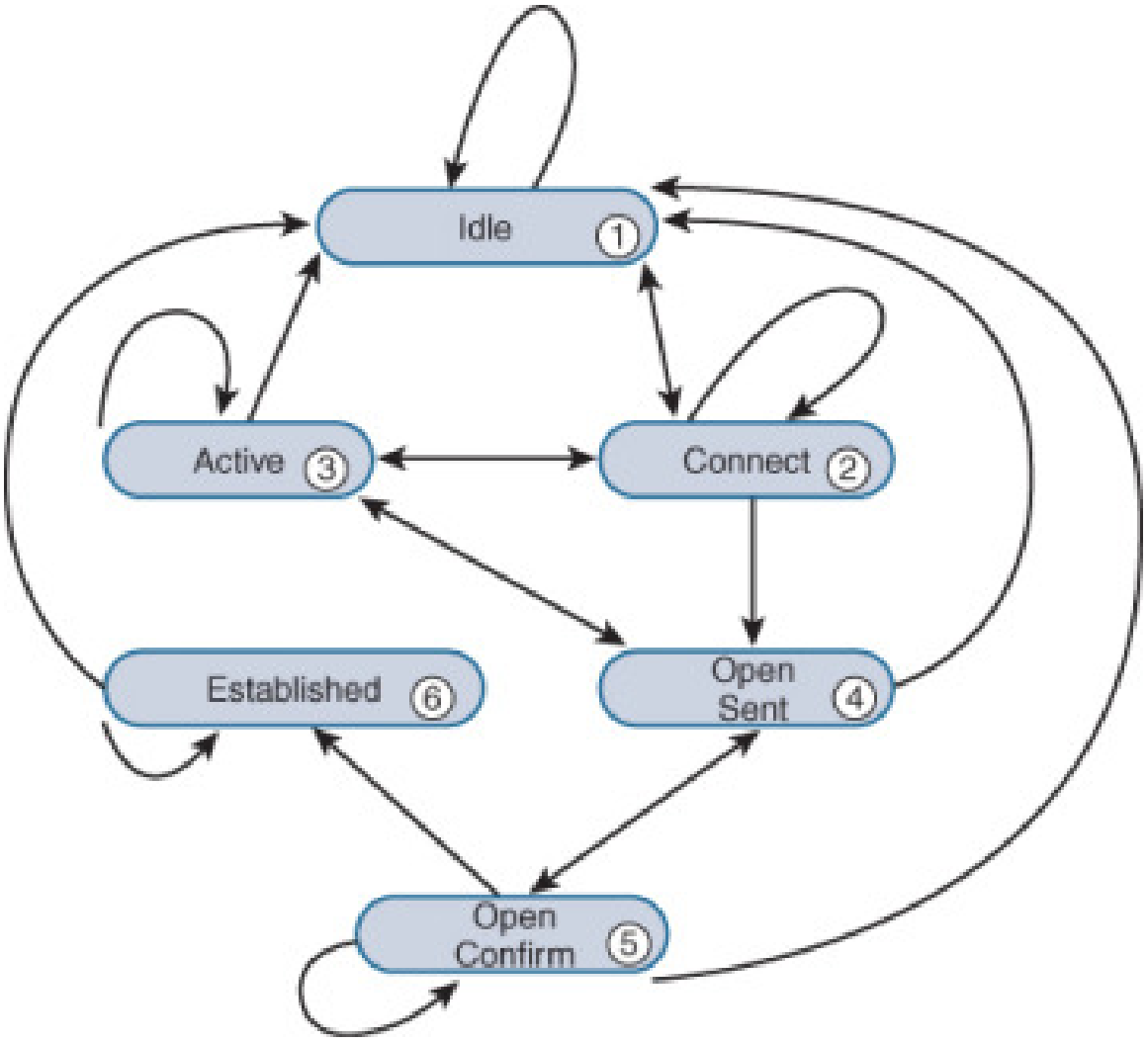
- **Successful Transition:** Upon receiving a KEEPALIVE from the peer, BGP transitions to the **Established** state.

- **Failed Transition:** If the hold timer expires before a KEEPALIVE is received, or if a NOTIFICATION message is received from the peer, BGP transitions back to **Idle**.

6. **Established State**

- **Meaning:** This is the goal! Both BGP peers have successfully exchanged and accepted OPEN messages, and the initial KEEPALIVE exchange has confirmed the session.

- **Actions:** In this state, the peers can exchange UPDATE messages (to advertise or withdraw routes), KEEPALIVE messages (to maintain the session), and NOTIFICATION messages (if errors occur). ROUTE-REFRESH messages can also be exchanged if the capability was negotiated.



BGP Timers

Understanding BGP Timers

BGP utilizes several timers to manage peering sessions, control update advertisements, and maintain routing table consistency. These can be broadly categorized into basic session timers and more advanced operational timers.

---

1. Basic BGP Session Timers

These timers are fundamental for establishing and maintaining BGP peerings. They are negotiated during the OPEN message exchange.

- **Keepalive Timer:**
  - **Purpose:** To ensure the BGP peer is still alive and the session is active. If no BGP UPDATE messages are sent, a KEEPALIVE message is transmitted.
  - **Default Value: 60 seconds**. This means each 60 sec BGP peer sends KEEPALIVE if no other updates are pending.

- **Hold-Down Timer (Hold Time):**
  - **Purpose:** The maximum amount of time (in seconds) that a BGP router will wait to receive a KEEPALIVE or UPDATE message from its peer before declaring the peer down, terminating the BGP session, and invalidating routes learned from that peer.
  - **Default Value: 180 seconds** (typically 3 times the default Keepalive interval).
  - **Negotiation:** During the OPEN message exchange, peers advertise their configured Hold Times. The **lower** of the two proposed Hold Times is then used for the session by both peers, provided it meets any "Minimum Acceptable Hold Time" configured by the other peer. The Keepalive interval is typically set locally and is often configured to be one-third of the agreed-upon Hold Time.

- **Minimum Acceptable Hold Time:**
  - **Purpose:** This locally configured timer protects a router from being forced by a peer to use an overly aggressive (very low) Hold Time. If a peer proposes a Hold Time in its OPEN message that is *less than* the local router's configured "Minimum Acceptable Hold Time," the local router will reject the peering attempt by sending a NOTIFICATION message (Error: OPEN Message Error, Subcode: Unacceptable Hold Time).
  - **Use Case:** An ISP might set a minimum acceptable hold time (e.g., 20-30 seconds) to prevent a customer from configuring an extremely low hold time (e.g., 3 seconds with 1-second keepalives) that could cause excessive processing or session instability on the ISP's router.

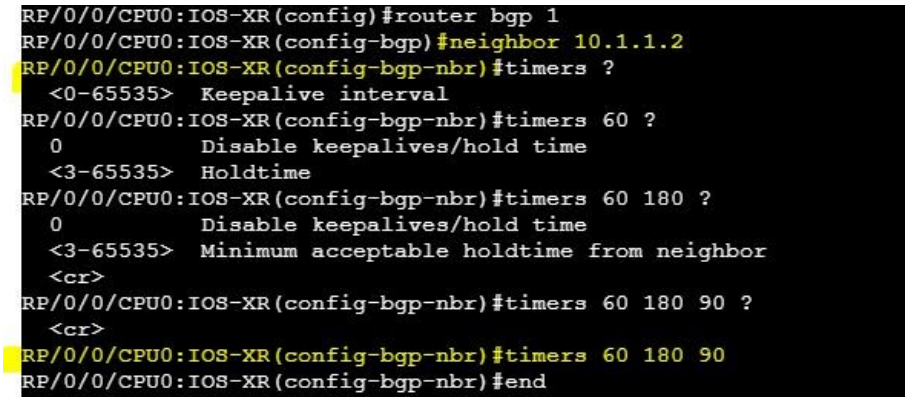
Changing Basic Timers:

- **IOS XE:**
  - View current timers: `show ip bgp neighbors <neighbor_ip>`
  - Configure:

```
R1(config)# router bgp <ASN>
R1(config-router)# neighbor <neighbor_ip> timers <keepalive_sec> <holdtime_sec> [<min_acceptable_holdtime_sec>]
```

- **IOS XR:**
  - View current timers: `show bgp neighbors <neighbor_ip>`
  - Configure (under neighbor configuration mode):

```
RP/0/RP0/CPU0:R1(config)# router bgp <ASN>
RP/0/RP0/CPU0:R1(config-bgp)# neighbor <neighbor_ip>
RP/0/RP0/CPU0:R1(config-bgp-neighbor)# timers <keepalive_sec> <holdtime_sec> [<min_acceptable_holdtime_sec>]
```



- **Important:** Changing these timers requires the BGP session to be reset (e.g., `clear ip bgp <neighbor_ip>` on XE or `clear bgp <neighbor_ip>` on XR) so that the new values can be negotiated in the OPEN messages. A soft reset or route refresh will not suffice for timer renegotiation.

---

2. Advanced BGP Operational Timers

These timers influence how BGP advertises routes and maintains the validity of paths.

- **Advertisement Interval Timer:**
  - **Purpose:** This timer controls the minimum delay between sending consecutive BGP UPDATE messages to a particular peer. It helps to pace BGP updates, preventing a router from flooding its peers with rapid updates, especially if many routes are changing or being added.
  - **Default Values:**
    - For **eBGP peers: 30 seconds**.
    - For **iBGP peers: 0 seconds** (meaning updates to iBGP peers can be sent immediately as they are processed, without an artificial delay between update batches).

- **Behavior:** If a router sends an UPDATE message to an eBGP peer, it will wait at least 30 seconds before sending another UPDATE message to that same peer, even if new routes or changes are processed internally before the 30-second interval is up. Updates can be batched.
- This timer can be viewed in the output of `show ip bgp neighbors <neighbor_ip>` (look for "Minimum advertisement interval").

- **BGP Scanner / Next-Hop Tracking:**

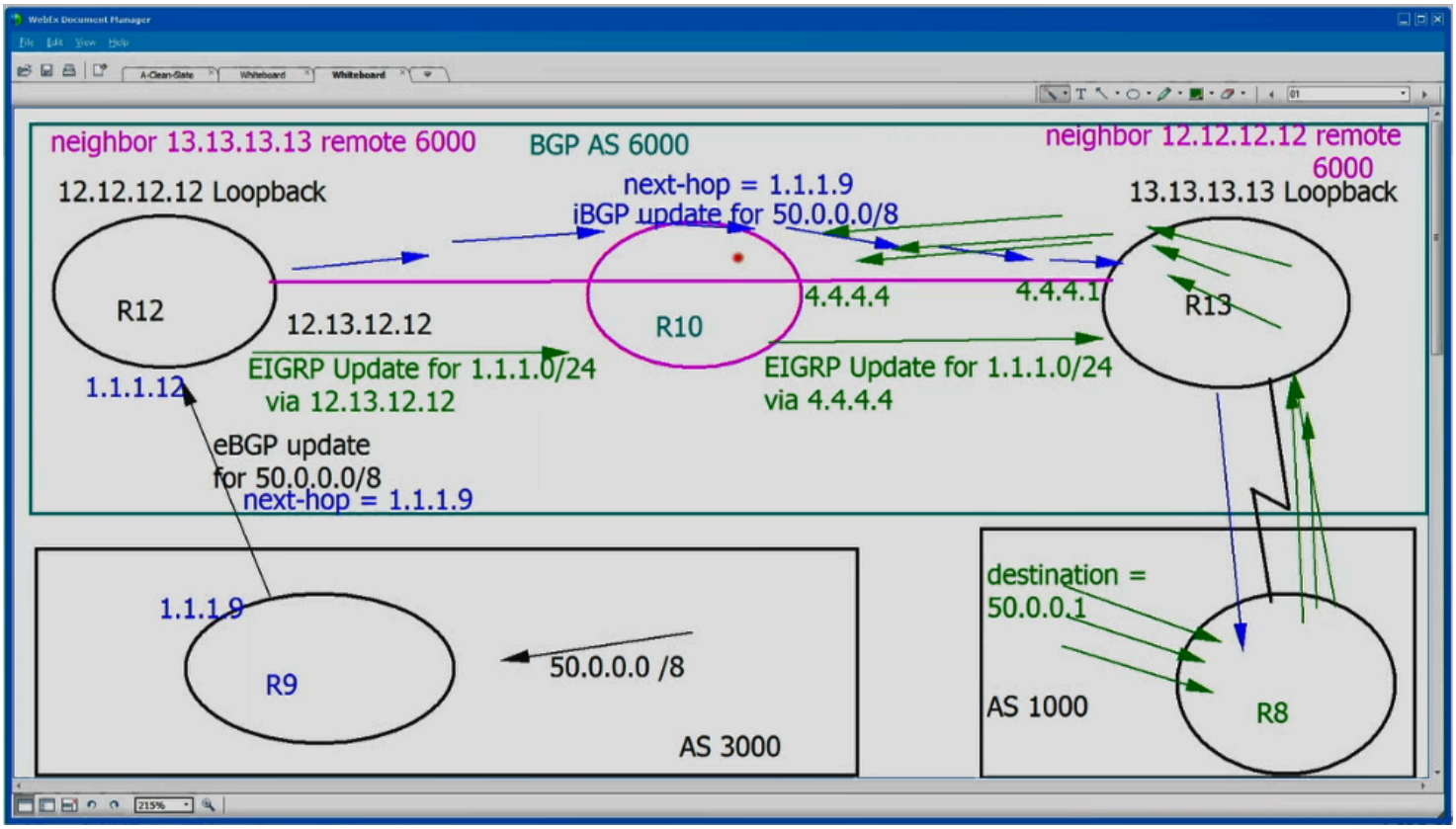
  - **BGP Scanner (Traditional):** Historically, BGP used a scanner process that would periodically (e.g., default **60 seconds**) walk through the BGP table to:
    - Validate the reachability of NEXT_HOP attributes for BGP routes (by checking the RIB for a valid IGP/static route to the next-hop). If a next-hop became unreachable, the BGP routes using it would be invalidated.
    - Perform other maintenance tasks like route dampening or conditional advertisement.

  - **Next-Hop Tracking (Modern):** The periodic BGP scanner for next-hop validation was relatively slow to react to underlying IGP topology changes. Modern BGP implementations use a more efficient, event-driven mechanism called **Next-Hop Tracking (NHT)**.
    - With NHT, BGP registers with the RIB for notifications about changes to the reachability of BGP next-hops. When the IGP detects a change that affects a BGP next-hop's reachability, the RIB informs BGP immediately.
    - This allows BGP to react much faster to next-hop failures or restorations, leading to quicker BGP convergence. NHT significantly enhances the role of the older, purely periodic BGP scanner for next-hop validation.

BGP Traffic Blackhole

When an Autonomous System (AS) acts as a transit path for BGP routes, ensuring all routers within that AS can correctly forward the traffic is crucial. If not all routers in the transit path have the necessary BGP information, traffic can be blackholed.

Blackhole Scenario Explained



1. **eBGP Route Learnt:** Router R9 (in AS 3000) advertises an external prefix (e.g., `50.0.0.0/8` with next-hop `1.1.1.9` - R9's IP) to R12 via eBGP.

2. **iBGP Propagation:** R12, being in the same AS as R13, advertises this route to R13 via iBGP. A key iBGP rule is that the **NEXT_HOP attribute is typically NOT changed** when advertising to an iBGP peer. So, R13 learns `50.0.0.0/8` with the next-hop still being `1.1.1.9` (R9).

3. **R10 is non-BGP Router**, which means it does not have any BGP configured

4. **eBGP Re-advertisement:** R13 then advertises this route to its eBGP peer R8 (in AS 1000). When doing so, R13 will change the next-hop to its own IP address facing R8.

5. **Traffic Ingress:** R8 receives traffic destined for `50.0.0.1` (within `50.0.0.0/8`) and forwards it to R13.

6. **R13's Forwarding Decision:**
    - R13 looks up `50.0.0.1`. It has a BGP route for `50.0.0.0/8` via next-hop `1.1.1.9`.
    - R13 now performs a **recursive lookup** to find how to reach `1.1.1.9`. R13 has an IGP route (e.g., EIGRP) to `1.1.1.9` via router R10.

7. **Packet to Non-BGP Router:** R13 forwards the packet destined for `50.0.0.1` to R10.

8. **The Blackhole at R10:** Router R10 receives the packet. However, R10 **is not running BGP**. It has no knowledge of the `50.0.0.0/8` prefix. It only knows its IGP routes. Since `50.0.0.1` is not in R10's IGP routing table, R10 has no specific route and **discards the packet**.

This is a traffic blackhole: R13 correctly believes it can reach `50.0.0.0/8` via `1.1.1.9`, and its IGP says `1.1.1.9` is via R10. But R10, the intermediate hop, doesn't know where `50.0.0.0/8` actually is.

Fixing the BGP Blackhole

Ensure All Transit Routers are BGP-Aware
The most straight forward way to solve this is to ensure that any router in the transit path for BGP traffic within your AS also participates in BGP.

1. **R10 Runs iBGP:** R10 should also be an iBGP peer with R12 and R13 (or with Route Reflectors). This way, R10 would also have the BGP route for `50.0.0.0/8` (next-hop `1.1.1.9`) and would know to forward packets for this prefix based on its IGP route to `1.1.1.9` (which would point towards R12).

2. **Scalable iBGP:** To avoid full-mesh iBGP in large ASes:
    - **Route Reflectors (RRs):** This is the most common solution. R10, R12, and R13 would be clients of one or more RRs.
    - **BGP Confederations:** Less common, divides the AS into sub-ASes.

Historical Solution: BGP Synchronization Rule

The BGP synchronization rule was designed to prevent this type of blackhole:

- **The Rule:** A BGP router will **not consider an iBGP-learned route as valid, will not use it, and will not advertise it to an eBGP peer UNLESS that same prefix is also learned via an IGP** (e.g., OSPF, IS-IS) running within the AS.

- **Intent:** To ensure that before an AS tells the outside world it can reach a prefix, all routers *within* the AS (via IGP) agree on how to reach it or the BGP next-hop for it.

- **The Problem with Synchronization:** To make synchronization work, you'd typically have to **redistribute BGP routes into your IGP**. This is highly problematic because:
    - IGPs are not designed to handle the massive number of routes in the BGP table.
    - Full redistribution can easily overwhelm the IGP, causing instability.

- **Default Status:** Because of these issues, BGP synchronization is **disabled by default** in most BGP Implementations.

- **IOS XR:** Cisco IOS XR **does not support** the BGP synchronization feature at all, as the design philosophy assumes either all transit path routers run BGP (likely with RRs) or MPLS is used to bypass the need for intermediate routers to have BGP routes.

Enabling/Disabling Synchronization (IOS XE):

```
R1(config)# router bgp <ASN>
R1(config-router)# synchronization      // To enable synchronization
R1(config-router)# no synchronization   // To disable synchronization (default)
```

```
IOS-XE#show ip bgp 50.50.50.0
BGP routing table entry for 50.50.50.0/24, version 3
Paths: (1 available, no best path)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    4.4.4.4 from 4.4.4.4 (4.4.4.4)
      Origin IGP, metric 0, localpref 100, valid, internal, not synchronized
      rx pathid: 0, tx pathid: 0
      Updated on May 23 2021 16:00:31 UTC
```

You can check if a route is not being used due to synchronization by looking at the output of `show ip bgp <prefix>`. If synchronization is enabled and the condition isn't met, the route might be in the BGP table but marked as "not synchronized" and not installed in the RIB as shown above.

Conclusion

While synchronization was an early attempt to solve the transit AS blackhole problem, the modern and preferred solutions involve:

1. Ensuring all routers in the transit path within an AS participate in iBGP (using Route Reflectors or Confederations for scalability).

2. Using MPLS, where core routers forward based on labels without needing to know the BGP routes. Relying on synchronization by redistributing BGP into an IGP is generally avoided due to its negative impact on IGP stability and performance.

BGP Next-hop

Next-Hop Behavior: IGPs vs. BGP

There's a key difference in how next-hops are handled when advertising routes:

1. **Best Route Advertisement:**

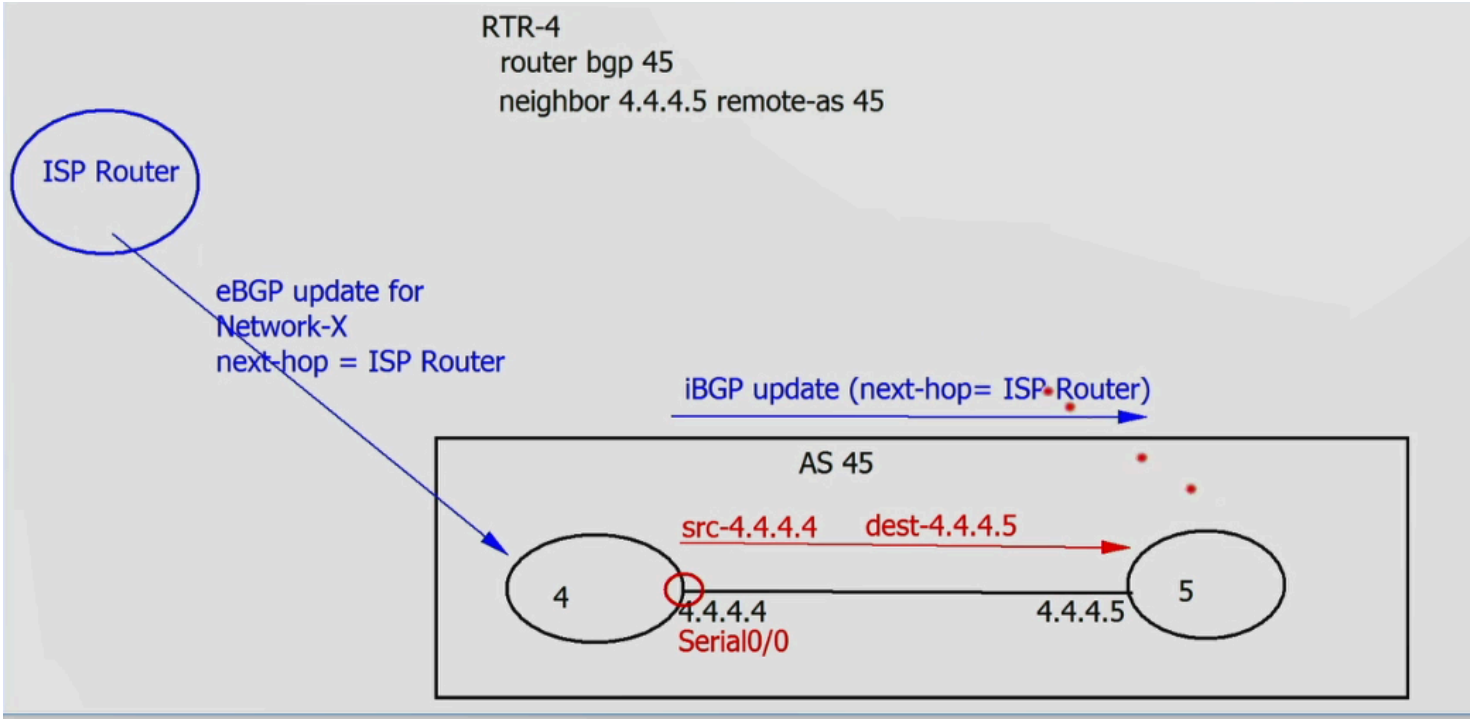
    - **IGPs & BGP:** Both protocol types generally aim to advertise only their best path to a destination to their neighbors/peers.
        - **IGPs:** While IGPs like OSPF and EIGRP can support Equal Cost Multi-Path (ECMP) in the RIB, when they advertise reachability (e.g., EIGRP updates or how OSPF/IS-IS describe links leading to prefixes), they are effectively guiding others to calculate the best path. Link-state protocols flood topology information, allowing each router to compute its own best path(s).
        - **BGP:** By default, BGP selects a single best path for each prefix based on its complex path attribute selection process and advertises only this best path.

2. **Next-Hop Advertisement:**

    - **IGPs:** When an IGP router advertises a route, the implicit or explicit next-hop is typically the advertising router itself if it's on the path.
    - **iBGP (Internal BGP):** This is where it differs significantly. When a router learns a route from an **eBGP (External BGP)** peer and then advertises that route to an **iBGP peer** (a peer within the same AS), iBGP has a specific rule: **the NEXT_HOP attribute is NOT changed by default.** The next-hop remains the IP address of the original eBGP peer from the external AS.

The iBGP Next-Hop Problem

This iBGP rule often leads to a reachability issue, as illustrated in this scenario:



1. **eBGP Learns Route:** Router 4 learns `Network-X` from the "ISP Router" via eBGP. The BGP `NEXT_HOP` attribute for `Network-X` is set to the IP address of the "ISP Router."

2. **iBGP Propagates Route:** Router 4 advertises `Network-X` to its iBGP peer, Router 5. Because of the iBGP rule, Router 4 sends the update with the `NEXT_HOP` attribute *still set to the IP address of the "ISP Router."*

3. **Router 5's:**
    - Router 5 receives the update for `Network-X` with the next-hop being the "ISP Router's" IP address.
    - **Problem:** Router 5 is an internal router in AS 45. It likely has no direct connection or route to the "ISP Router" in the external AS. Its IGP knows about routers *within* AS 45, but not about the link between ISP & R4.
    - **Consequence:**
        - Router 5 cannot resolve the BGP next-hop for `Network-X`.
        - Because the next-hop is unreachable, Router 5 **will not install `Network-X` into its RIB (Routing Information Base)**, even though it's in its BGP table.
        - Since the route isn't in the RIB as a valid best path, Router 5 will not advertise `Network-X` to any of its other BGP peers (e.g., another eBGP peer).
        - If you issue `show ip bgp <Network-X>` on Router 5, you'd likely see the route as "inaccessible" due to an unreachable next-hop.

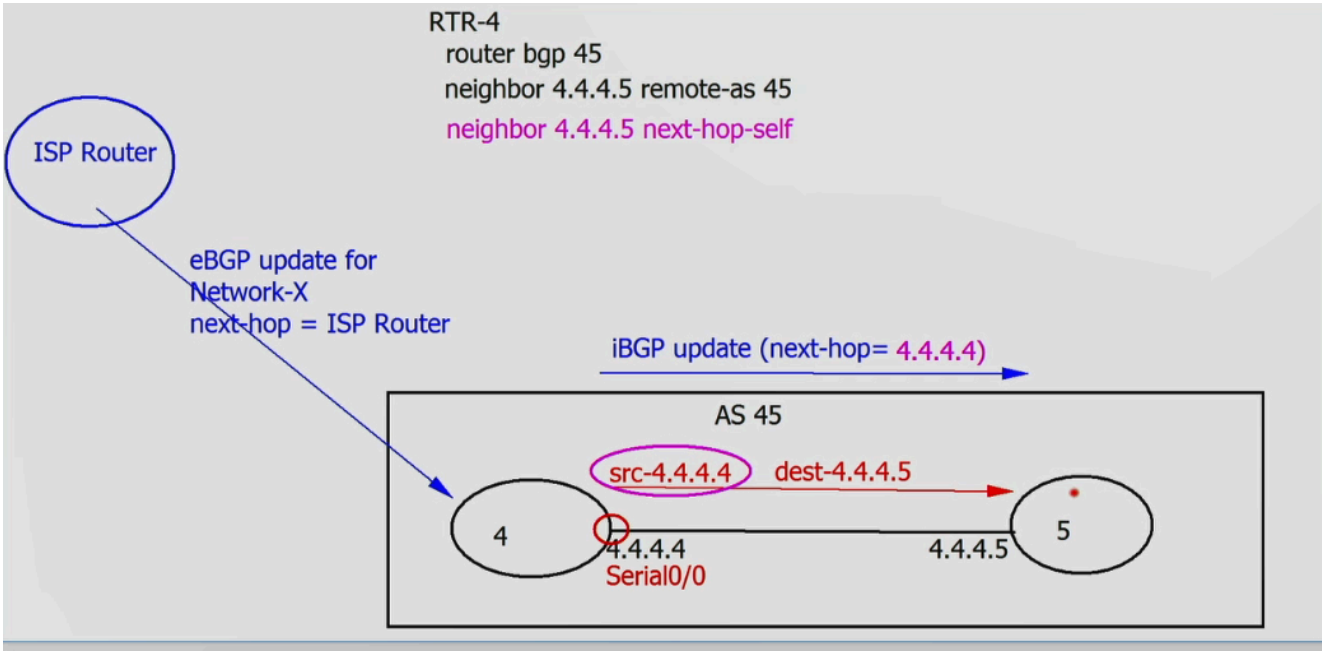
This effectively means that while R4 learned the route, it cannot be properly used by or propagated through R5 unless R5 can reach the original eBGP next-hop.

Solving the iBGP Next-Hop Issue

There are two main ways to solve this:

1. **Make the External Next-Hop Reachable via IGP:**
    - One could theoretically redistribute the network connecting Router 4 to the "ISP Router" (which contains the next-hop IP) into the IGP of AS 45.
    - If Router 5 then learns a route to the "ISP Router's" IP address via its IGP, it could resolve the next-hop.
    - **Drawbacks**
        - It leaks external infrastructure details into your IGP.
        - If the link between R4 and the ISP Router goes down, the IGP would need to reconverge, and the static redistribution of a connected route might not accurately reflect actual reachability if not done carefully.

2. **Use `next-hop-self`:**
    - This BGP command is configured on Router 4 (the one peering with both the eBGP source and the iBGP destination).
    - When `next-hop-self` is applied to the iBGP peering with Router 5, Router 4 will **change the NEXT_HOP attribute to its own IP address** when advertising eBGP-learned routes to Router 5.



- **Configuration:**
  - **IOS XE (Classic BGP Config Mode):**

```
R4(config)# router bgp 45
R4(config-router)# neighbor <R5_iBGP_PEER_IP> next-hop-self
```

  - **IOS XE (Address Family Config Mode):**

```
R4(config)# router bgp 45
R4(config-router)# neighbor <R5_iBGP_PEER_IP> remote-as 45
R4(config-router)# address-family ipv4 unicast
R4(config-router-af)# neighbor <R5_iBGP_PEER_IP> next-hop-self
```

  - **IOS XR:**

```
RP/0/0/CPU0:R4(config)# router bgp 45
RP/0/0/CPU0:R4(config-bgp)# neighbor <R5_iBGP_PEER_IP>
RP/0/0/CPU0:R4(config-bgp-nbr)# remote-as 45
RP/0/0/CPU0:R4(config-bgp-nbr)# address-family ipv4 unicast
RP/0/0/CPU0:R4(config-bgp-nbr-af)# next-hop-self
```

- **Result:**
  - Router 5 receives the update for `Network-X` with the next-hop being Router 4's IP address.

- Since Router 4 and Router 5 are in the same AS and presumably have IGP reachability to each other's peering addresses, Router 5 can resolve Router 4 as the next-hop.
- Router 5 installs `Network-X` into its RIB and can now advertise it to other BGP peers if needed.

Note: The iBGP next-hop issue primarily applies to routes learned from eBGP peers and then advertised to iBGP peers. When advertising to **eBGP peers**, the next-hop is **changed to the advertising router's IP by default.**

BGP Routes Advertisement

Injecting and Advertising Routes in BGP

Unlike IGPs where route advertisement often happens automatically for interfaces participating in the protocol, BGP requires explicit configuration to inject routes into its process before they can be advertised to peers. Once a BGP router has valid routes in its BGP table (and in its RIB), it will then advertise these best paths to its neighbors, subject to policy and BGP's own propagation rules.

---

1. Using the `network` Command

This command is used to **originate** a prefix directly into the BGP table. BGP will then advertise this prefix to its peers.

- **How it Works:**
  - You specify a network and mask using the `network` command within the BGP process.
  - BGP checks the router's global Routing table (RIB) for an **exact match** of this prefix (network and mask).
  - If an exact match exists in the RIB (learned via any source such as connected, static, or an IGP), BGP installs this prefix into its BGP table with itself as the originator.
  - The BGP path attributes for the route will typically include:
    - **ORIGIN code:** `i` (IGP) - indicating it was originated via the `network` command from the AS.
    - **NEXT_HOP:** `0.0.0.0` in the local BGP table, which is then changed to the router's appropriate update-source IP when advertising to a peer.

- **Critical Points:**
  - The prefix **must exist in the RIB** for BGP to originate it via the `network` command.
  - The `network` command does **not** enable BGP on an interface (unlike how it often works in IGPs).
  - It's solely for injecting existing RIB prefixes into the BGP process.

- **Configuration:**
  - **IOS XE (Classic):** `R1(config-router)# network X.X.X.X [mask M.M.M.M]` *(If the mask is omitted, BGP assumes a classful mask and looks for that classful network in the RIB.)*
  - **IOS XE (Address Family):**

```
R1(config-router)# address-family ipv4 unicast
R1(config-router-af)# network x.x.x.x mask m.m.m.m
```

  - **IOS XR:**

```
RP/0/0/CPU0:IOS-XR(config)# router bgp <ASN>
RP/0/0/CPU0:IOS-XR(config-bgp)# address-family ipv4 unicast
RP/0/0/CPU0:IOS-XR(config-bgp-af)# network X.X.X.X/L  // Or network X.X.X.X mask M.M.M.M
```

- **Advertising Aggregates with `network` command:** A common technique for advertising a summary route is to create a static route for the aggregate pointing to Null0 (to ensure it's always in the RIB), and then use the `network` command to inject this aggregate into BGP (this is useful if you have many prefixes, so instead of typing one by one manually just do this approach)

```
R1(config)# ip route 185.30.0.0 255.255.252.0 Null0 // For 185.30.0.0/22
R1(config)# router bgp <ASN>
R1(config-router)# network 185.30.0.0 mask 255.255.252.0
```

---

2. Using the `redistribute` Command

This command injects routes learned from other routing sources (like IGPs, connected interfaces, or static routes) into the BGP table.

- **Why Use Redistribution?** It's a practical way to inject many prefixes into BGP without manually configuring a `network` command for each one, especially when these prefixes are already managed by an IGP.

- **Origin Code:** Routes redistributed into BGP typically receive an **ORIGIN code of `?` (Incomplete)**. In BGP best path selection, Origin code `i` (IGP, from `network` command) is preferred over `e` (EGP, obsolete), which is preferred over `?` (Incomplete).

- **Metric (MED):** When redistributing from an IGP into BGP, BGP often takes the IGP's metric and places it into the **MED (Multi-Exit Discriminator)** BGP attribute by default. This behavior can be influenced by the `default-metric` command under BGP if the IGP metric isn't directly translatable or if specific handling is needed. Unlike OSPF redistribution (which often uses a default metric of 20 type 2), BGP doesn't have a fixed default "BGP metric" it applies to the path in the same way for redistributed routes; its preference is driven by attributes.

- **Configuration:**
  - **IOS XE (Classic):** `R1(config-router)# redistribute <protocol> [process-id] [options]`
  - **IOS XE (Address Family):**

    ```
    R1(config-router)# address-family ipv4 unicast
    R1(config-router-af)# redistribute <protocol> [process-id] [options]
    ```

  - **IOS XR:**

    ```
    RP/0/0/CPU0:IOS-XR(config)# router bgp <ASN>
    RP/0/0/CPU0:IOS-XR(config-bgp)# address-family ipv4 unicast
    RP/0/0/CPU0:IOS-XR(config-bgp-af)# redistribute <protocol> [process-id_or_instance-name] [route-policy <POLICY_NAME>]
    ```

---

3. Using Route Summarization

While not a primary way to *inject* routes from outside BGP, the `aggregate-address` command creates a new summary (aggregate) route in the BGP table and advertises it.

- **How it Works:** This command looks for **more specific routes that already exist in the BGP table** (learned via `network` command, redistribution, or from other BGP peers) that fall within the specified aggregate range.

- **Prerequisite:** The more specific "child" routes must be present in the BGP table for the aggregate to be generated and advertised.

- **Advertisement:** By default, BGP advertises both the aggregate route and the more specific routes. The `summary-only` option can be used to advertise only the aggregate and suppress the specifics.

- **Path Attributes:** The aggregate route inherits its path attributes from the contributing specific routes (e.g., AS_PATH can become an AS_SET). Options exist to modify these attributes (e.g., `as-set`, `attribute-map`).

- **Configuration:**
  - **IOS XE (Classic):** `R1(config-router)# aggregate-address <prefix> <mask> [summary-only] [options]`
  - **IOS XE (Address Family):**

```
R1(config-router)# address-family ipv4 unicast
R1(config-router-af)# aggregate-address <prefix> <mask> [summary-only] [options]
```

  - **IOS XR:**

```
RP/0/RP0/CPU0:IOS-XR(config)# router bgp <ASN>
RP/0/RP0/CPU0:IOS-XR(config-bgp)# address-family ipv4 unicast
RP/0/RP0/CPU0:IOS-XR(config-bgp-af)# aggregate-address <prefix/length> [summary-only] [options]
```

---

4. Advertising Prefixes Received from Other BGP Peers

This is a fundamental behavior of BGP.

- When a BGP router receives route updates from a peer, it processes these routes through its inbound policies, runs its best path selection algorithm, and installs the best valid paths into its BGP table and potentially its RIB.

- These selected best paths are then **automatically eligible for advertisement to other BGP peers** (both iBGP and eBGP), subject to:
  - Outbound routing policies.
  - iBGP split-horizon rule (a route learned from an iBGP peer is not advertised to another iBGP peer).
  - Next-hop processing rules (e.g., eBGP changes next-hop to self; iBGP does not by default).

Policy Control (IOS XR):

IOS XR, by default, has a "deny all" policy for routes advertised to eBGP peers. You must create and apply an outbound route policy (even if it's a simple "permit all" policy) to an eBGP neighbor session for any routes to be advertised.

- **Example "Permit All" Policy (IOS XR):**

```
route-policy PASS_ALL
  pass
end-policy
!
router bgp <ASN>
  neighbor <NEIGHBOR_IP>
    address-family ipv4 unicast
      route-policy PASS_ALL out // Apply policy outbound
```

This explicit policy requirement in IOS XR promotes safer BGP deployments by forcing administrators to define what they intend to advertise.

These methods provide the mechanisms to populate a BGP router's table and share reachability information with the rest of the BGP-speaking world.

BGP Default Route Advertisement

how to advertise a default route (`0.0.0.0/0`) to your BGP peers. This is common for ISPs providing internet access to customers or for enterprises guiding traffic towards an internet gateway.

There are primarily two distinct approaches:

---

1. Advertise Default Route to a Specific Neighbor

This method uses a per-neighbor configuration to send a default route *only* to that designated peer.

- **How it Works:** The `default-originate` command (on the neighbor or neighbor address-family configuration) instructs the BGP router to originate and advertise a `0.0.0.0/0` route to that specific neighbor.
    - **Unconditional Advertisement:** By default, this command advertises the default route regardless of whether `0.0.0.0/0` exists in the router's own RIB.
    - **Conditional Advertisement:** You can append a `route-map` (IOS XE) or `route-policy` (IOS XR). The default route will then only be advertised to the neighbor if the conditions in the route-map/policy are met (e.g., if a specific prefix exists in the RIB, signaling that the router has valid upstream connectivity).
- **Key Point:** This command **does not require the default route to be present in the router's main routing table (RIB)** to be advertised unconditionally to the specified neighbor. It *generates* the default route advertisement for that peer.

Configuration:

- **IOS XE (Classic & Address Family):**

```
R1(config)# router bgp <ASN>
! Classic
R1(config-router)# neighbor <NEIGHBOR_IP> default-originate [route-map <MAP_NAME>]

! Address Family
R1(config-router)# address-family ipv4 unicast
R1(config-router-af)# neighbor <NEIGHBOR_IP> default-originate [route-map <MAP_NAME>]
```

- **IOS XR:**

```
RP/0/0/CPU0:R1(config)# router bgp <ASN>
RP/0/0/CPU0:R1(config-bgp)# neighbor <NEIGHBOR_IP>
RP/0/0/CPU0:R1(config-bgp-nbr)# address-family ipv4 unicast
RP/0/0/CPU0:R1(config-bgp-nbr-af)# default-originate [route-policy <POLICY_NAME>]
```

---

2. Advertise Default Route to All BGP Peers

If you want to advertise a default route more broadly from your BGP process to all or many peers (subject to normal outbound policies), you first need to get the `0.0.0.0/0` route into your BGP table as a locally originated prefix. There are a couple of ways to achieve this:

A. Using the network Command:

This is often the cleanest way.

1. **Ensure `0.0.0.0/0` is in the RIB:** Create a static route for `0.0.0.0/0` pointing to `Null0`. This ensures the route exists in the RIB for BGP to pick up and also prevents routing loops for the default route on this router.

    - **IOS XE:** `R1(config)# ip route 0.0.0.0 0.0.0.0 Null0`
    - **IOS XR:**

    ```
    RP/0/0/CPU0:R1(config)# router static
    RP/0/0/CPU0:R1(config-static)# address-family ipv4 unicast
    RP/0/0/CPU0:R1(config-static-af)# 0.0.0.0/0 Null0
    ```

2. **Originate with `network` command:**

    - **IOS XE:**

    ```
    R1(config)# router bgp <ASN>
    R1(config-router)# network 0.0.0.0
    ! Or, more explicitly:
    ! R1(config-router-af)# address-family ipv4 unicast
    ! R1(config-router-af)# network 0.0.0.0 mask 0.0.0.0
    ```

    - **IOS XR:**

    ```
    RP/0/0/CPU0:R1(config)# router bgp <ASN>
    RP/0/0/CPU0:R1(config-bgp)# address-family ipv4 unicast
    RP/0/0/CPU0:R1(config-bgp-af)# network 0.0.0.0/0
    ```

Once injected via the `network` command, this default route will be advertised to all eBGP and iBGP peers.

B. Using Redistribution:

1. **Ensure `0.0.0.0/0` is in the RIB:** Same as above, create a static route to `Null0`.

2. **Redistribute Static Routes:**

    - **IOS XE:**

```
R1(config)# router bgp <ASN>
R1(config-router)# address-family ipv4 unicast // Or configure directly under router bgp
R1(config-router-af)# redistribute static route-map ONLY_DEFAULT
!
R1(config)# route-map ONLY_DEFAULT permit 10
R1(config-route-map)# match ip address prefix-list DEFAULT_PREFIX
!
R1(config)# ip prefix-list DEFAULT_PREFIX seq 5 permit 0.0.0.0/0
```

C. Using `default-information originate`

- IOS XR has a `default-information originate` command directly under `router bgp <ASN>` or `router bgp <ASN> address-family ipv4 unicast`.

- This command **unconditionally** originates a default route and advertises it to all neighbors within that BGP instance/address family. It does **not** require the default route to already exist in the RIB via static or other means (though having a Null0 static route for it locally is still good practice for loop prevention on the originating router).

```
RP/0/0/CPU0:R1(config)# router bgp <ASN>
RP/0/0/CPU0:R1(config-bgp)# address-family ipv4 unicast // Or configure globally
RP/0/0/CPU0:R1(config-bgp-af)# default-information originate [route-policy <POLICY_NAME>]
```

If a `route-policy` is attached, the default route origination can be made conditional.

CCIE #66356

BGP AD Value

Changing BGP Administrative Distances

Administrative Distance (AD) is the first criteria router uses to determine the trustworthiness of a routing source if it learns about the same prefix from multiple routing protocols. A lower AD is more preferred.

By default, BGP routes have the following ADs:

- **eBGP (External BGP) routes:** 20
- **iBGP (Internal BGP) routes:** 200
- **Locally originated BGP routes** (e.g., via `network` or `aggregate-address`): 200

You can modify these default ADs.

---

1. Changing AD Based on BGP Route Type

You can set new AD value for all eBGP routes, all iBGP routes, and all locally originated BGP routes.

- **IOS XE (Classic and Address Family):**

```
R1(config)# router bgp <YOUR_ASN>
! For Classic CLI:
R1(config-router)# distance bgp <external_AD> <internal_AD> <local_AD>
! Example: distance bgp 25 205 205

! For Address Family CLI (applies within the specific AF):
R1(config-router)# address-family ipv4 unicast
R1(config-router-af)# distance bgp <external_AD> <internal_AD> <local_AD>
```

  - `<external_AD>`: New AD for routes learned from eBGP peers (default 20).
  - `<internal_AD>`: New AD for routes learned from iBGP peers (default 200).
  - `<local_AD>`: New AD for routes originated locally by this BGP (default 200).

- **IOS XR:**

```
RP/0/RP0/CPU0:R1(config)# router bgp <YOUR_ASN>
RP/0/RP0/CPU0:R1(config-bgp)# address-family ipv4 unicast
RP/0/RP0/CPU0:R1(config-bgp-af)# distance bgp <external_AD> <internal_AD> <local_AD>
```

---

2. Changing AD for Specific Routes (Per-Prefix/Per-Neighbor)

Sometimes you need more granular control to change the AD only for specific prefixes, perhaps learned from a particular neighbor.

- **IOS XE:**

This is achieved using the distance <AD_value> <neighbor_IP> <neighbor_wildcard> <access-list_number_or_name> command.

1. **Create a Standard Access Control List (ACL)** to match the *network prefixes* for which you want to change the AD.
   - When a standard ACL is used in this BGP `distance` command, it matches network prefixes.
     - `access-list 1 permit host 10.1.1.0` would match only the prefix `10.1.1.0/32`.
     - `access-list 1 permit 10.1.0.0 0.0.255.255` would match any prefix starting with `10.1.x.x`.

```
! Example: To match prefix 192.168.1.0/24
R1(config)# access-list 10 permit 192.168.1.0 0.0.0.255
! Example: To match host route 10.10.10.10/32
R1(config)# access-list 11 permit host 10.10.10.10
```

2. **Apply the new AD within the BGP process:**

```
R1(config)# router bgp <YOUR_ASN>
! For Classic CLI:
R1(config-router)# distance <NEW_AD> <NEIGHBOR_IP> <NEIGHBOR_WILDCARD> <ACL_NUMBER_OR_NAME>
! Example: Set AD to 25 for routes matched by ACL 10, learned from neighbor 1.1.1.1
! R1(config-router)# distance 25 1.1.1.1 0.0.0.0 10

! For Address Family CLI:
R1(config-router)# address-family ipv4 unicast
R1(config-router-af)# distance <NEW_AD> <NEIGHBOR_IP> <NEIGHBOR_WILDCARD> <ACL_NUMBER_OR_NAME>
```

   - `<NEW_AD>`: The administrative distance (1-255) you want to assign.
   - `<NEIGHBOR_IP>`: The IP address of the BGP peer from which these routes are learned.
   - `<NEIGHBOR_WILDCARD>`: A wildcard mask applied to the neighbor IP. `0.0.0.0` matches that specific neighbor.
   - `<ACL_NUMBER_OR_NAME>`: The standard ACL that identifies the prefixes.

- **IOS XR:**

1. **Define a Prefix-Set** (or an IPv4 ACL) to match the routes:

```
RP/0/RP0/CPU0:R1(config)# prefix-set MY_SPECIFIC_PREFIXES
RP/0/RP0/CPU0:R1(config-pfx)# 192.168.1.0/24,
RP/0/RP0/CPU0:R1(config-pfx)# 10.10.10.10/32
RP/0/RP0/CPU0:R1(config-pfx)# end-set
```

2. **Create a Route Policy** to set the desired AD for these matched prefixes:

```
RP/0/RP0/CPU0:R1(config)# route-policy SET_CUSTOM_AD
RP/0/RP0/CPU0:R1(config-rpl)# if destination in MY_SPECIFIC_PREFIXES then
RP/0/RP0/CPU0:R1(config-rpl-if)# set distance 25
RP/0/RP0/CPU0:R1(config-rpl-if)# endif
RP/0/RP0/CPU0:R1(config-rpl)# pass // Important: allow other routes to pass with default AD
RP/0/RP0/CPU0:R1(config-rpl)# end-policy
```

3. **Apply the Route Policy** to the BGP neighbor (inbound) or globally within the address family:

```
RP/0/RP0/CPU0:R1(config)# router bgp <YOUR_ASN>
RP/0/RP0/CPU0:R1(config-bgp)# neighbor <NEIGHBOR_IP>
RP/0/RP0/CPU0:R1(config-bgp-nbr)# address-family ipv4 unicast
RP/0/RP0/CPU0:R1(config-bgp-nbr-af)# route-policy SET_CUSTOM_AD in // Apply to routes learned from this neighbor
```

BGP Table

Understanding the BGP Table (`show ip bgp`)

Similar to how OSPF/IS-IS has their Link-State Database (LSDB) and EIGRP has its Topology Table, BGP maintains its own **BGP Table**. This table stores:

- Routes **locally originated** by this router (e.g., via the `network` command or redistribution).
- Routes **received as BGP UPDATE messages** from BGP peers.

After populating this BGP table, the BGP process runs its **Best Path Selection Algorithm** against all paths available for each prefix. Only the selected best paths are then considered for installation into the global IP routing table and for advertisement to other BGP peers.

Common Commands to View BGP Information

- **Overall BGP Table:**
  - **IOS XE:** `R1# show ip bgp`
  - **IOS XR:** `R1# show bgp ipv4 unicast`

- **Peer-Specific Route Information (IOS XE/XR):**
  - `show bgp [ipv4 unicast] neighbors <neighbor_ip> routes`: Shows routes learned from this neighbor that have passed inbound policies and are eligible for best-path selection.
  - `show bgp [ipv4 unicast] neighbors <neighbor_ip> advertised-routes`: Shows routes that this router is advertising *to* this neighbor (after outbound policies).
  - `show bgp [ipv4 unicast] neighbors <neighbor_ip> received-routes`:
    - Historically, to see the *raw, unfiltered* routes received from a peer before inbound policy evaluation, `soft-reconfiguration inbound` needed to be configured for that peer.
    - Without `soft-reconfiguration inbound`, this command typically shows routes received from the peer *that have been accepted by any inbound route policy*. Modern BGP implementations often rely on route refresh capabilities rather than storing a full unfiltered set.

- **BGP Peering Summary:**
  - **IOS XE:** `R1# show ip bgp summary`
  - **IOS XR:** `R1# show bgp summary` or `show bgp ipv4 unicast summary`

Decoding the `show ip bgp` Output

```
RouterA# show ip bgp
BGP table version is 14, local router ID is 172.31.11.1
Status codes: s suppressed, d damped, h history, * valid, > best, i -
internal, r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete
   Network          Next Hop            Metric LocPrf Weight Path
*> 10.1.0.0/24      0.0.0.0                  0         32768 i
* i                 10.1.0.2                 0    100      0 i
*> 10.1.1.0/24      0.0.0.0                  0         32768 i
*>i10.1.2.0/24      10.1.0.2                 0    100      0 i
*> 10.97.97.0/24    172.31.1.3                             0 64998 64997 i
*                   172.31.11.4                            0 64999 64997 i
* i                 172.31.11.4              0    100      0 64999 64997 i
*> 10.254.0.0/24    172.31.1.3               0             0 64998 i
*                   172.31.11.4                            0 64999 64998 i
* i                 172.31.1.3               0    100      0 64998 i
r> 172.31.1.0/24    172.31.1.3               0             0 64998 i
r                   172.31.11.4                            0 64999 64998 i
r i                 172.31.1.3               0    100      0 64998 i
*> 172.31.2.0/24    172.31.1.3               0             0 64998 i
<output omitted>
```

Let's break down the columns:

1. **Status Codes (First characters before the Network):**
    - `*` (asterisk): Indicates that the path is **valid**. This means BGP has performed basic checks (e.g., the next-hop is resolvable, AS_PATH is loop-free for eBGP) and considers this path eligible for best-path calculation.
    - `>` (greater than): Indicates this is the **best path** selected by the BGP decision process for this prefix. This is the path that BGP will attempt to install into the IP routing table. Only one path per prefix can be the best path by default.
    - `r` (RIB-Failure): The path is valid from BGP's perspective (`*` is usually also present), but it **could not be installed into the IP routing table (RIB)**. Common reasons include:
        - A route to the same prefix from another routing protocol with a better (lower) Administrative Distance already exists.
        - The BGP next-hop, while known to BGP, is not properly resolvable in the RIB via an IGP or static route.
        - Other RIB installation policies.

- `i`: The route was learned from an **iBGP peer**. If absent for a learned route, it was learned from an eBGP peer.

- Other codes exist (e.g., `s` for suppressed, `d` for dampened, `h` for history, `S` for Stale in GR).

2. **Network:**
   - The IP prefix (e.g., `10.1.0.0/24`).

3. **Next Hop:**
   - The IP address of the next-hop router to reach this prefix.

   - If this is `0.0.0.0`, it signifies that the prefix was **locally originated** on this router (e.g., via the `network` command or `aggregate-address`). When advertising this route, the router will change the next-hop to its own appropriate IP address.

   - For routes learned from eBGP peers, this is the IP address of the eBGP peer.

   - For routes learned from iBGP peers, this is typically the original eBGP next-hop (unless `next-hop-self` was used by the advertising iBGP peer).

4. **Metric (MED - Multi-Exit Discriminator):**
   - An optional non-transitive attribute used to influence path selection among different entry points into a neighboring AS. Lower is generally better (More about this later in Path Attributes section)

5. **LocPrf (Local Preference):**
   - Used within an AS to influence the preferred exit point from the AS for a given prefix. Higher is better. Typically only exchanged between iBGP peers. (More about this later in Path Attributes section)

6. **Weight:**
   - A Cisco-proprietary attribute, local to the router. It's the first attribute checked in the BGP best path selection. Higher weight is preferred. Default for learned routes is 0; for locally originated routes, it's 32768.

7. **Path (AS_Path):**

- The sequence of Autonomous System numbers the route has traversed to reach this router. Read from **left to right**:
  - The **leftmost AS** is the AS of the neighbor that advertised this path to your router.
  - The **rightmost AS** is the originating AS of the prefix.

8. **Origin Code (Last character on the line):**
   - Indicates how the prefix was originally injected into BGP:
     - `i` - **IGP:** Originated via a `network` command within an AS. This is the most preferred origin.
     - `e` - **EGP:** Originated from the historic EGP protocol (obsolete). Less preferred than `i`.
     - `?` - **Incomplete:** Origin is unknown. Typically means the route was redistributed into BGP from another routing protocol (like an IGP or static routes), or it's an aggregate route where the origin of all components couldn't be determined consistently. Least preferred origin.

Once a prefix is in the BGP table, and it's considered valid (*) and best (>), it will be a candidate for installation into the IP routing table. Furthermore, this best path will typically be advertised to other BGP peers (both iBGP and eBGP), subject to outbound policies and BGP propagation rules (like iBGP split horizon). You do not need a separate network command for these BGP-learned routes to be re-advertised.

BGP Path Attributes

Understanding BGP Path Attributes

BGP Path Attributes are pieces of information that describe the characteristics of a path to a destination prefix. When a BGP router generates an UPDATE message to advertise a prefix (e.g., `1.0.0.0/8`), it includes a set of these attributes. These attributes provide details about the route, such as its origin, the sequence of Autonomous Systems (ASes) it has traversed, its next-hop IP address, and many other policy-related characteristics.

Why Do We Need BGP Path Attributes?

1. **Best Path Selection:** The primary use of path attributes is in the **BGP Best Path Selection Algorithm**. When a BGP router learns multiple paths to the same destination prefix from different peers, it uses a specific sequence of evaluating these attributes to choose the single best path. This best path is then installed in the router's BGP table (and potentially its RIB) and advertised to other BGP peers.
    - *Note:* Not all attributes are directly used in every step of the standard best-path algorithm, but many of the key ones are.

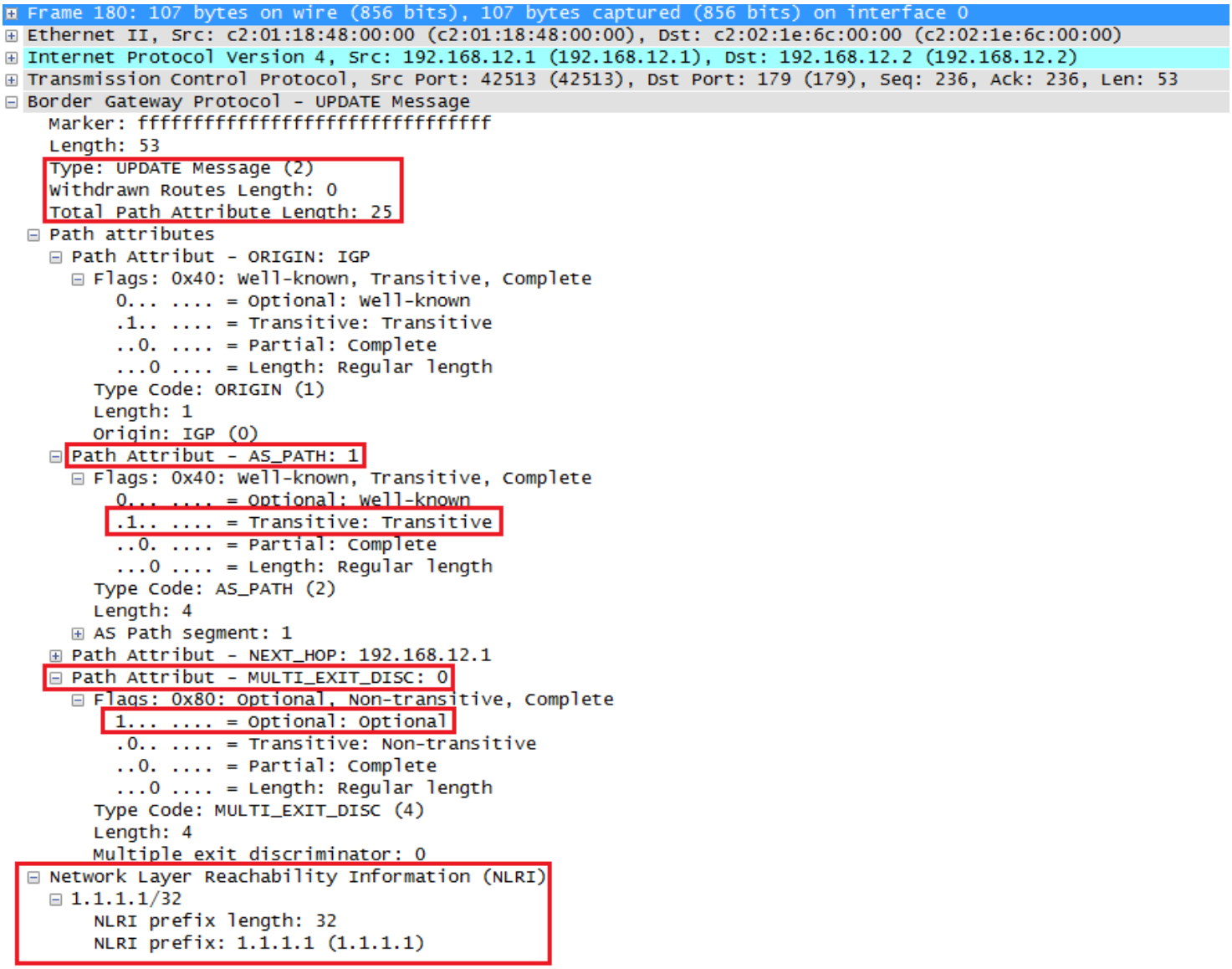
2. **Loop Prevention:** Some attributes are fundamental for protocol operation, such as the **AS_PATH attribute**, which is the primary mechanism for detecting and preventing inter-AS routing loops.

3. **Policy Implementation:** Attributes like LOCAL_PREF, MED, Communities, and Extended Communities allow network administrators to implement complex routing policies, influencing how traffic enters or exits their AS.

4. **Supporting Advanced Features:** Many attributes enable advanced BGP functionalities like MPLS VPNs (e.g., Extended Communities, MP_REACH_NLRI), IPv6 support (MP_REACH_NLRI, AS4_PATH), Route Reflection (e.g., ORIGINATOR_ID, CLUSTER_LIST), and more.

Path Attribute Structure in an UPDATE Packet

Each path attribute is encoded in a **Type-Length-Value (TLV)-like format** within a BGP UPDATE message. More precisely, the structure is:



Each BGP Path Attribute, before you get to its specific Type Code and Value, starts with a one-octet (1-byte) Attribute Flags field. This byte provides critical handling instructions.

Here's a breakdown of the bits within that Attribute Flags octet, starting from the most significant bit (MSB):

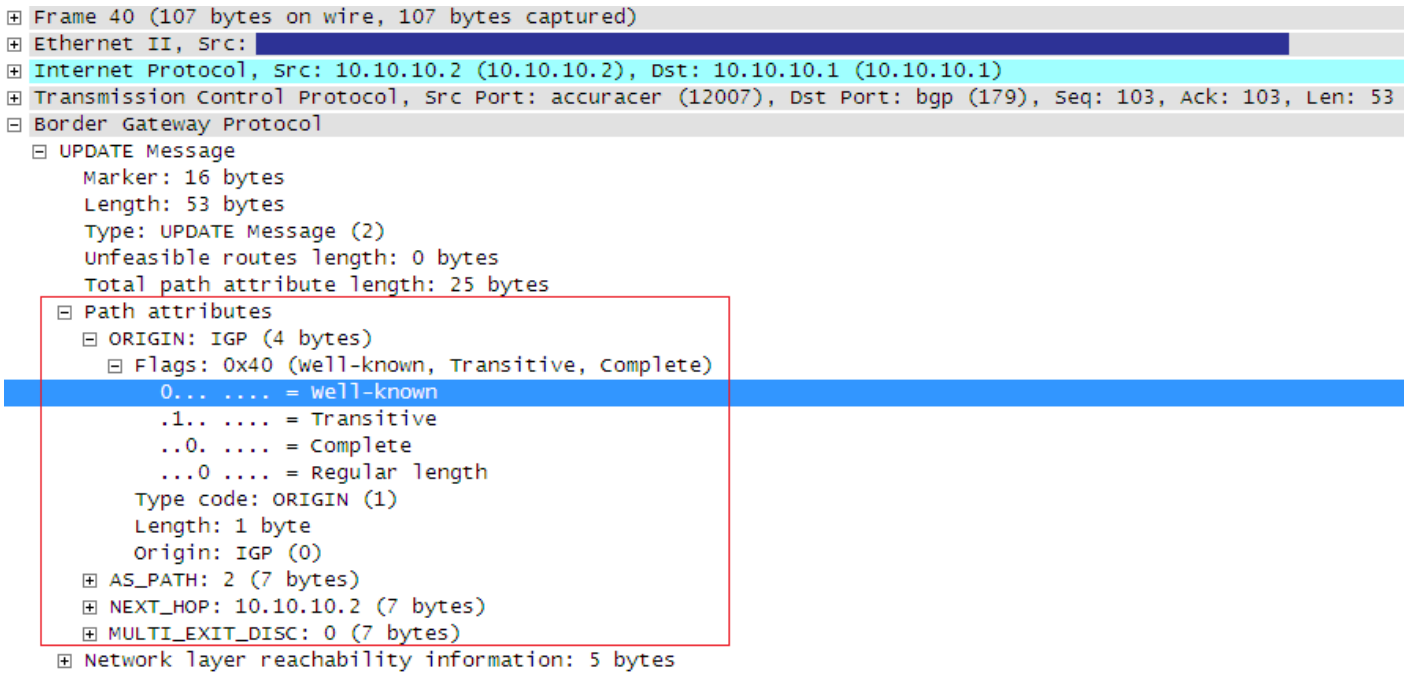
```
Bit:    0         1          2         3         |  4   5   6   7
      +--------+----------+---------+----------+-----------------+
      |Optional|Transitive| Partial | Extended |    (Unused)     |
      |  Bit   |   Bit    |   Bit   |  Length  |                 |
      +--------+----------+---------+----------+-----------------+
```

1. Optional Bit (Bit 0 - Most Significant Bit)

- **Value 0: Attribute is Well-Known**

- **Meaning:** This attribute type is fundamental to BGP and **must be recognized by all BGP implementations**.

- **Value 1: Attribute is Optional**
  - **Meaning:** This attribute type is an extension to BGP and **is not required to be recognized or supported by all BGP implementations**. Many advanced features or newer capabilities are introduced as Optional attributes.
  - **Consequence:** If a BGP speaker receives an Optional attribute it doesn't recognize, its behavior is determined by the **Transitive Bit** (Bit 1). It does not send a NOTIFICATION message solely for an unrecognized optional attribute.



2. Transitive Bit (Bit 1)

This bit's meaning is primarily tied to how unrecognized **Optional** attributes are handled.

- **For Well-Known Attributes (Optional Bit = 0):**
  - The Transitive Bit **MUST be set to 1** (Transitive). All Well-Known attributes are inherently transitive. They are expected to be understood and propagated correctly by all BGP speakers.

- **For Optional Attributes (Optional Bit = 1):**
  - **Value 1: Optional Transitive Attribute**
    - **Meaning:** If a BGP speaker receives an Optional attribute with the Transitive bit set to 1, but it *does not recognize* this specific attribute type, it **should accept the attribute, mark it as partial (see Partial Bit below), and pass it along unchanged to its other BGP peers.**

        - **Purpose:** This allows new Optional Transitive attributes to traverse BGP speakers that haven't yet been upgraded to understand them. This ensures that the attribute can reach other parts of the network where it might be understood and utilized.

    - **Value 0: Optional Non-Transitive Attribute**
        - **Meaning:** If a BGP speaker receives an Optional attribute with the Transitive bit set to 0, and it *does not recognize* this specific attribute type, it **must quietly ignore and discard the attribute**. It should **not** pass this unrecognized attribute on to its other BGP peers.
        - **Purpose:** These attributes are typically meant for local significance within an AS or between directly peering ASes that understand them, and are not intended for wider propagation if unrecognized.

3. Partial Bit (Bit 2)

This bit is relevant primarily for **Optional Transitive** attributes.

- **Value 1: Information is Partial**
    - **Meaning:** This bit is set to 1 by a BGP speaker when it forwards an **Optional Transitive** attribute that it **does not recognize**.
    - **Purpose:** It serves as an indication to downstream BGP speakers that the attribute has traversed at least one router that did not understand it. While the attribute's value is preserved, the "partial" flag suggests that its integrity or completeness from the perspective of the originating router's full intent might not be fully verifiable by routers that *do* understand the attribute later in the path, because an intermediate router didn't process it.

- **Value 0: Information is Complete**
    - **Meaning:** For Well-Known attributes and Optional Non-Transitive attributes, the Partial bit **MUST be set to 0**. For an Optional Transitive attribute that *is* recognized by the BGP speaker forwarding it (or being originated by it), the Partial bit is also set to 0.

4. Extended Length Bit (Bit 3)

This bit indicates the size of the "Attribute Length" field that follows the Attribute Type Code.

- **Value 0: Attribute Length is One Octet**
    - **Meaning:** The "Attribute Length" field (which specifies the length of the "Attribute Value") is 1 byte long. This means the attribute value itself can be up to 255 bytes long.

- **Value 1: Attribute Length is Two Octets**
    - **Meaning:** The "Attribute Length" field is 2 bytes long. This allows the attribute value to be longer than 255 bytes (up to 65,535 bytes).

- **Purpose:** Necessary for attributes that might carry a large amount of data (e.g., MP_REACH_NLRI with many prefixes, or complex community attributes).

Unused Bits (Bits 4-7)

- These bits are currently unused.
- They **MUST be set to zero** by the sender and **should be ignored** by the receiver. This allows for future extensions if needed.

Categorization of BGP Path Attributes

BGP attributes are categorized based on two main criteria:

1. Based on Implementation Requirements:

- **Well-Known Attributes:** These attributes **must be recognized by all BGP implementations**.
  - **Well-Known Mandatory:** These attributes **must be present** in all UPDATE messages for prefixes they logically apply to (e.g., ORIGIN, AS_PATH, NEXT_HOP).
  - **Well-Known Discretionary:** These attributes must be recognized by all BGP implementations but **may or may not be sent** in a particular UPDATE message (e.g., LOCAL_PREF, ATOMIC_AGGREGATE).
- **Optional Attributes:** These attributes are **not required to be supported by all BGP implementations**.

2. Based on Scope or (Inter-AS Propagation):

- **Transitive Attributes:**
  - If a BGP speaker receives an UPDATE message with an **Optional Transitive** attribute that it does not understand, it should preserve the attribute, mark it as "partial" (by setting the Partial bit in the flags), and pass it along to its other BGP peers.
  - All Well-Known attributes are inherently transitive.
- **Non-Transitive Attributes:**
  - If a BGP speaker receives an UPDATE message with an **Optional Non-Transitive** attribute that it does not understand, it **must quietly ignore and discard the attribute** and not pass it on to other peers.

Overview List of Key BGP Path Attributes

Here's a look at some of the most important and commonly encountered attributes from your list:

Type Code	Name	Category	Transitivity Notes	Reference
1	**ORIGIN**	Well-Known Mandatory	Transitive	RFC 4271
2	**AS_PATH**	Well-Known Mandatory	Transitive	RFC 4271
3	**NEXT_HOP**	Well-Known Mandatory	Transitive (for a given address family)	RFC 4271
4	**MULTI_EXIT_DISC (MED)**	Optional	Non-Transitive (not advertised to eBGP peers in another AS)	RFC 4271
5	**LOCAL_PREF**	Well-Known Discretionary	Transitive *within an AS and to confederation peers*; **NOT sent to external eBGP peers**.	RFC 4271
6	**ATOMIC_AGGREGATE**	Well-Known Discretionary	Transitive	RFC 4271
7	**AGGREGATOR**	Optional	Transitive	RFC 4271
8	**COMMUNITY**	Optional	Transitive	RFC 1997
9	**ORIGINATOR_ID**	Optional	Non-Transitive (used with Route Reflectors)	RFC 4456
10	**CLUSTER_LIST**	Optional	Non-Transitive (used with Route Reflectors)	RFC 4456
14	**MP_REACH_NLRI**	Optional	Non-Transitive	RFC 4760
15	**MP_UNREACH_NLRI**	Optional	Non-Transitive	RFC 4760
16	**Extended Communities**	Optional	Transitive (often used for VPNs, EVPNs)	RFC 4360
17	**AS4_PATH**	Optional	Transitive (for 4-octet ASNs when peer doesn't support new AS_PATH with 4-octet ASNs)	RFC 6793
18	**AS4_AGGREGATOR**	Optional	Transitive (for 4-octet ASNs)	RFC 6793
32	**LARGE_COMMUNITY**	Optional	Transitive	RFC 8092

Summary Points on Path Attributes:

1. **Diverse Usage:** Path attributes serve many functions beyond just best-path selection, including loop prevention, policy enforcement, and enabling advanced services like MPLS VPNs, Multicast

VPNs, and Segment Routing.

2. **Type Code Range:** Attribute Type Codes are 1 byte (0-255). Code 0 and 255 are reserved, leaving up to 254 potential attribute types. Many of these are unassigned or reserved for future use.

3. **Well-Known Attributes & Transitivity:**
    - Well-Known Mandatory attributes (ORIGIN, AS_PATH, NEXT_HOP) are transitive.
    - Well-Known Discretionary attributes (LOCAL_PREF, ATOMIC_AGGREGATE): ATOMIC_AGGREGATE is transitive. LOCAL_PREF is transitive within the local AS (including across confederation sub-ASes) but is **not** advertised to external eBGP peers.

4. **Optional Attributes:** Can be either Transitive or Non-Transitive. If an Optional Transitive attribute is unrecognized, it's passed on (marked as partial). If an Optional Non-Transitive attribute is unrecognized, it's dropped.

5. **Cisco Proprietary "Weight":** It's important to remember the **Weight** attribute, which is Cisco proprietary. It is the very first attribute checked in the Cisco BGP best-path algorithm, is locally significant to the router, and is **not transmitted** to BGP peers.

BGP Weight

BGP Path Attribute: Weight (Cisco Proprietary)

While many BGP path attributes are defined in RFCs and exchanged between BGP peers, **Weight** is a Cisco-proprietary parameter.

What is BGP Weight?

- **Local Significance:** Weight is a value assigned to BGP paths that is **only significant to the local router** where it is configured. It is **not advertised to or received from BGP peers** in BGP UPDATE messages.

- **Path Selection Influence:** Its primary purpose is to influence the BGP best path selection process on the local router. When a router has multiple paths to the same destination prefix, it will prefer the path with the **highest Weight**.

- **Cisco Only:** Being Cisco proprietary, it's only effective on Cisco routers and won't be understood or used by routers from other vendors.

Weight Values

- **Routes Learned from BGP Peers:** By default, routes learned from any BGP neighbor (eBGP or iBGP) are assigned a Weight of **0**.

- **Locally Originated Routes:** Routes originated by the local router (e.g., using the `network` command, redistribution, or `aggregate-address`) are assigned a default Weight of **32,768**.

- **Range:** Weight is a 16-bit value, ranging from **0 to 65,535**.

- **Preference:** A **higher** Weight value is more preferred.

HINT

The default assignments of 0 and 32,768 cannot be changed; However the weight for any path can be modified using configuration commands.

How to Change BGP Weight

1. Setting Weight for All Prefixes from a Specific Neighbor:

This assigns a specific weight to all routes learned from a particular BGP peer.

- **IOS XE (Classic & Address Family):**

```
R1(config)# router bgp <YOUR_ASN>
! For Classic CLI:
R1(config-router)# neighbor <NEIGHBOR_IP> weight <0-65535>

! For Address Family CLI (applies within the specific AF):
R1(config-router)# address-family ipv4 unicast
R1(config-router-af)# neighbor <NEIGHBOR_IP> weight <0-65535>
```

- **IOS XR:**

```
RP/0/RP0/CPU0:R1(config)# router bgp <YOUR_ASN>
RP/0/RP0/CPU0:R1(config-bgp)# neighbor <NEIGHBOR_IP>
RP/0/RP0/CPU0:R1(config-bgp-nbr)# address-family ipv4 unicast
RP/0/RP0/CPU0:R1(config-bgp-nbr-af)# weight <0-65535>
```

2. Setting Weight for Specific Prefixes (using Route-Map/RPL):

This is the most common method for granular control, typically applied to routes received from a neighbor (inbound policy).

- **IOS XE (using Route-Map):**
  1. **Create an Access Control List (ACL) or Prefix-List** to match the desired prefixes:

```
R1(config)# access-list 10 permit 11.11.11.0 0.0.0.255 // Matches 11.11.11.0/24
! Or using a prefix-list
R1(config)# ip prefix-list PL_MY_PREFIXES permit 11.11.11.0/24
```

  2. **Create a Route-Map** to match the ACL/prefix-list and set the weight:

```
R1(config)# route-map SET_WEIGHT_RM permit 10
R1(config-route-map)# match ip address 10 // or match ip address prefix-list PL_MY_PREFIXES
R1(config-route-map)# set weight 1234
R1(config-route-map)# exit
R1(config)# route-map SET_WEIGHT_RM permit 20 // Implicitly permits other routes without changing weight
```

  3. **Apply the Route-Map to the BGP Neighbor (inbound):**

```
R1(config)# router bgp <YOUR_ASN>
! For Classic CLI:
R1(config-router)# neighbor <NEIGHBOR_IP> route-map SET_WEIGHT_RM in
! For Address Family CLI:
R1(config-router)# address-family ipv4 unicast
R1(config-router-af)# neighbor <NEIGHBOR_IP> route-map SET_WEIGHT_RM in
```

- **IOS XR (using Route Policy Language - RPL):**
  1. **Define a Prefix-Set** (optional, can match directly in RPL) and a **Route Policy**:

```
RP/0/RP0/CPU0:R1(config)# prefix-set MY_PREFIXES_FOR_WEIGHT
RP/0/RP0/CPU0:R1(config-pfx)# 11.11.11.0/24
RP/0/RP0/CPU0:R1(config-pfx)# end-set
!
RP/0/RP0/CPU0:R1(config)# route-policy SET_WEIGHT_RPL
RP/0/RP0/CPU0:R1(config-rpl)# if destination in MY_PREFIXES_FOR_WEIGHT then
RP/0/RP0/CPU0:R1(config-rpl-if)# set weight 1234
RP/0/RP0/CPU0:R1(config-rpl-if)# else
RP/0/RP0/CPU0:R1(config-rpl-else)# pass // Allow other routes
RP/0/RP0/CPU0:R1(config-rpl-else)# endif
RP/0/RP0/CPU0:R1(config-rpl)# end-policy
```

2. **Apply the Route Policy to the BGP Neighbor (inbound):**

```
RP/0/RP0/CPU0:R1(config)# router bgp <YOUR_ASN>
RP/0/RP0/CPU0:R1(config-bgp)# neighbor <NEIGHBOR_IP>
RP/0/RP0/CPU0:R1(config-bgp-nbr)# address-family ipv4 unicast
RP/0/RP0/CPU0:R1(config-bgp-nbr-af)# route-policy SET_WEIGHT_RPL in
```

3. Setting Weight Based on AS_PATH (IOS XE):

You can influence the weight of routes learned from a neighbor based on their AS_PATH.

- **IOS XE:**

1. Define an `ip as-path access-list` to match specific AS_PATH patterns using regular expressions.

```
R1(config)# ip as-path access-list 10 permit ^65001_ // Matches paths originating from AS 65001
```

2. Apply a `filter-list` with a weight to the neighbor.

```
R1(config)# router bgp <YOUR_ASN>
R1(config-router)# neighbor <NEIGHBOR_IP> filter-list 10 weight 500
```

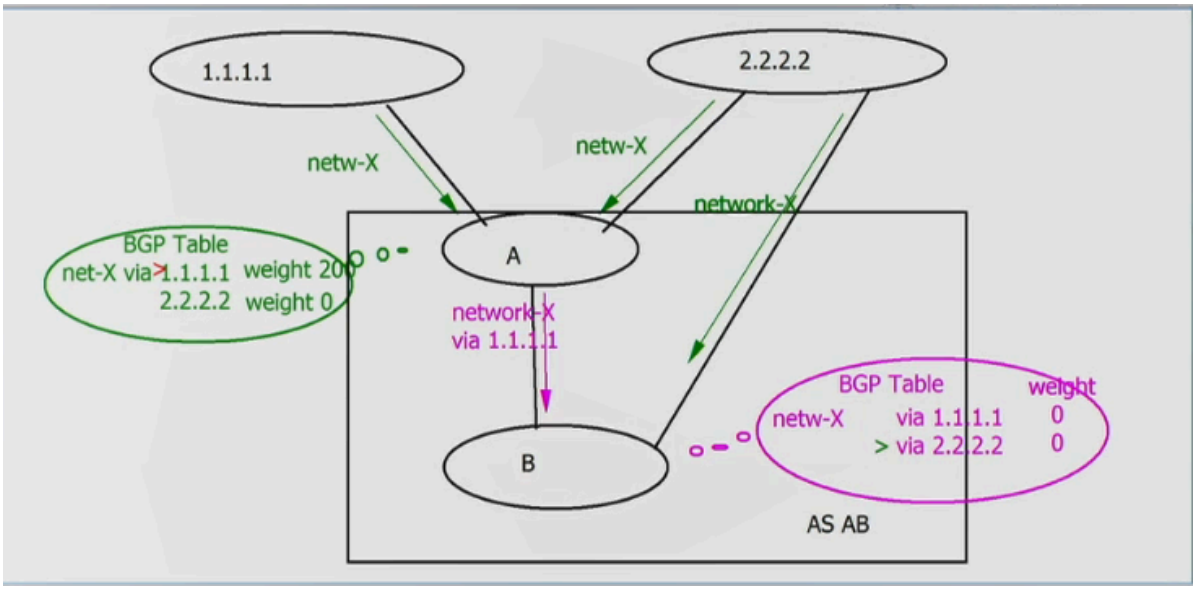
This sets a weight of 500 for routes learned from `<NEIGHBOR_IP>` whose AS_PATH matches AS_PATH ACL 10.

- **IOS XR:** This type of AS_PATH based weight setting would also be done using RPL, matching on `as-path in (AS_PATH_SET_NAME)` and then using `set weight`.

Weight in the BGP Best Path Selection Algorithm

Weight is the very first attribute checked in Cisco's BGP best path selection process.

1. **Prefer path with the highest Weight.** If weights are different for multiple paths to the same prefix, BGP will choose the one with the highest weight, and further path attributes are not considered for those lower-weight paths.



Verifying Weight

You can see the weight assigned to BGP paths using:

- **IOS XE:**
  - `show ip bgp`
  - `show ip bgp <prefix>`

- **IOS XR:**
  - `show bgp ipv4 unicast`
  - `show bgp ipv4 unicast <prefix>`

Use Cases

1. **Influencing Local Exit Point:**

   - If a border router in an enterprise or ISP has connections to multiple upstream providers or peers and learns the same prefixes from them, Weight can be used to make one provider's path an absolute preference *on that router*. The path with the highest weight will be chosen by that router, and this decision happens before other attributes like LOCAL_PREF or AS_PATH length are even considered.

2. **Simple Primary/Backup Configuration:**

- For an enterprise router connected to two ISPs (ISP A - primary, ISP B - backup), an administrator can set a higher weight for all routes learned from ISP A and a lower weight (or leave default 0) for routes learned from ISP B. This makes ISP A the primary exit path on that router.

3. **Influencing Path Selection within an AS (via iBGP):**

   - While iBGP peers do not see the weight value set on an upstream router, the *decision made* by the upstream router (based on weight) influences which path is advertised to iBGP peers. If Router A prefers a path via ISP1 due to a high weight setting, Router A will advertise only that path (its best path) to its iBGP neighbor Router B. Router B then acts based on the path it receives, without knowing weight was the deciding factor on Router A.

BGP Local Preference

BGP Path Attribute: Local Preference

Local Preference is BGP path attribute used to influence outbound routing decisions **within your own Autonomous System (AS)**.

What is Local Preference?

- **Definition:** Local Preference is a **Well-Known Discretionary** attribute (Type Code 5).
  - **Well-Known:** All BGP implementations must recognize it.
  - **Discretionary:** It may or may not be included in every BGP UPDATE message.

- **Scope and Transitivity:**
  - It is **always included in iBGP (Internal BGP) UPDATE messages** sent between peers within the same AS.
  - It is **NOT included in eBGP (External BGP) UPDATE messages** sent to peers in different Autonomous Systems (an exception is between sub-ASes within a BGP Confederation).
  - Despite being classified with the "transitive" bit set in its attribute flags (as all Well-Known attributes are), its propagation is limited to the local AS or confederation.

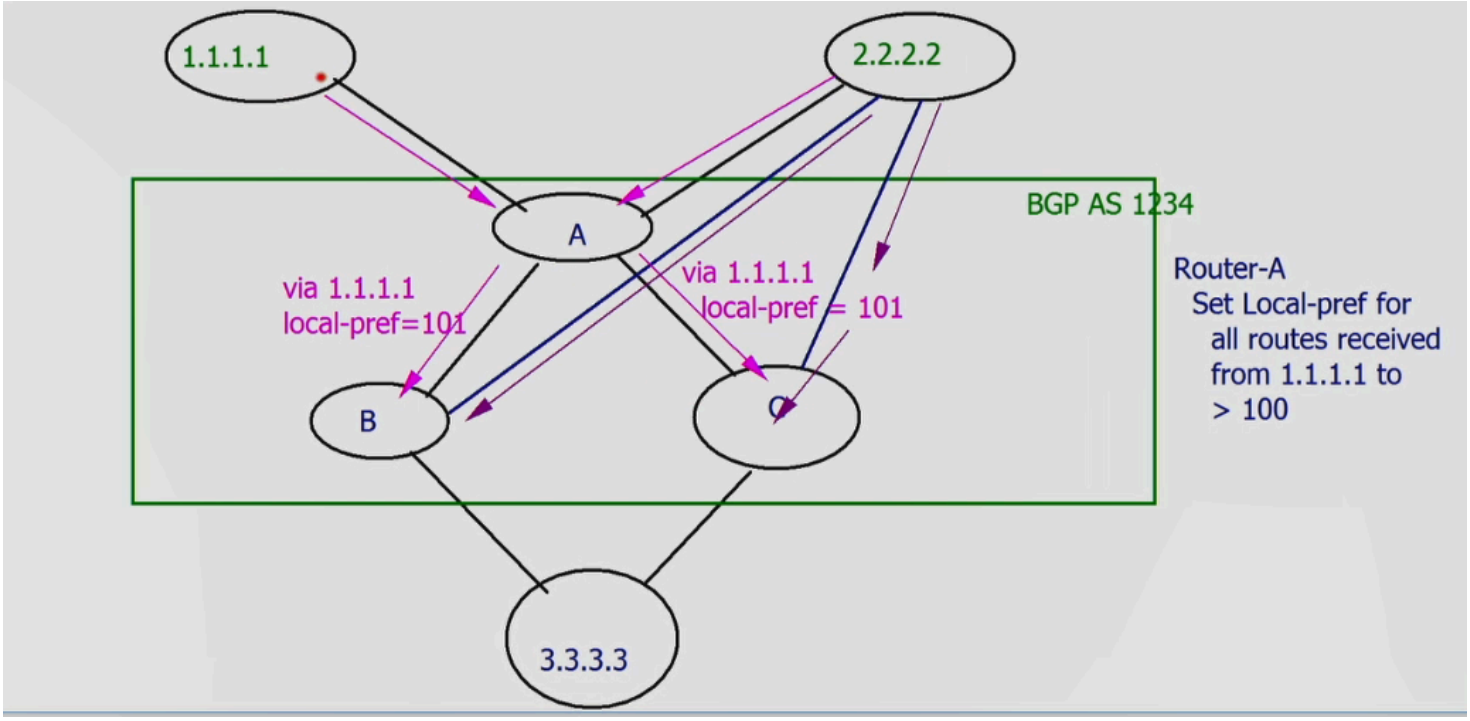
Why Use Local Preference?

Local Preference provides a consistent way to tell all routers *within your AS* which **exit point (or which eBGP peer connection) to prefer** when sending traffic to an external destination.

- **AS-Wide Outbound Policy:** If your AS has multiple exit points to the internet (e.g., connections to two different ISPs), you can use Local Preference to designate one ISP as the primary exit and the other as a backup for all or specific prefixes.

- **Example:** An enterprise network is connected to ISP-A and ISP-B. They want all internal routers to prefer using ISP-A for outbound internet traffic.
  - On the enterprise edge router(s) connected to ISP-A, you would set a higher Local Preference for routes learned from ISP-A.
  - This higher Local Preference value is then advertised via iBGP to all other iBGP speakers within the enterprise AS.

- All internal iBGP routers will then prefer the path that has the higher Local Preference, ensuring consistent selection of the exit path via ISP-A.



Local Preference Value

- **Data Type:** It's an unsigned 32-bit integer (allowing values up to over 4 billion).
- **Preference:** A **higher** Local Preference value is more preferred.
- **Default Value:**
  - Routes learned from an **eBGP peer** are assigned a default Local Preference of **100** by the router that receives them (before being advertised to iBGP peers).
  - Routes **locally originated** on a router (e.g., via `network` or `aggregate-address` commands) also get a default Local Preference of **100**.
  - This default value of 100 **can be changed globally** on a router.

How to Change Local Preference

1. Change the Global Default Local Preference:

This changes the default Local Preference value (from 100) that the router assigns to routes it originates or learns from eBGP peers before any policy.

- **IOS XE (Classic & Address Family CLI):**

```
R1(config)# router bgp <YOUR_ASN>
R1(config-router)# bgp default local-preference <0-4294967295>
```

- **IOS XR:**

```
RP/0/RP0/CPU0:R1(config)# router bgp <YOUR_ASN>
RP/0/RP0/CPU0:R1(config-bgp)# bgp default local-preference <0-4294967295>
```

2. Set Local Preference for Specific Routes (Using Route-Map/RPL):

This is the most common method, typically applied inbound on eBGP sessions to set a preference for routes as they enter your AS. This new Local Preference is then advertised to your iBGP peers.

- **IOS XE (using Route-Map):**

    1. Define an ACL or Prefix-List to match prefixes:

```
R1(config)# ip prefix-list PL_FROM_ISP_A permit 55.55.55.0/24
```

    2. Create a Route-Map to set Local Preference:

```
R1(config)# route-map SET_LP_ISP_A permit 10
R1(config-route-map)# match ip address prefix-list PL_FROM_ISP_A
R1(config-route-map)# set local-preference 200
R1(config-route-map)# exit
R1(config)# route-map SET_LP_ISP_A permit 20 // To permit other routes without changing LP
```

    3. Apply to the eBGP neighbor (inbound):

```
R1(config)# router bgp <YOUR_ASN>
R1(config-router)# neighbor <ISP_A_PEER_IP> route-map SET_LP_ISP_A in
```

- **IOS XR (using Route Policy Language - RPL):**

    1. Define a Prefix-Set (optional) and a Route Policy:

```
RP/0/RP0/CPU0:R1(config)# prefix-set PS_FROM_ISP_A
RP/0/RP0/CPU0:R1(config-pfx)# 55.55.55.0/24
RP/0/RP0/CPU0:R1(config-pfx)# end-set
!
RP/0/RP0/CPU0:R1(config)# route-policy SET_LP_ISP_A_RPL
RP/0/RP0/CPU0:R1(config-rpl)# if destination in PS_FROM_ISP_A then
RP/0/RP0/CPU0:R1(config-rpl-if)# set local-preference 200
RP/0/RP0/CPU0:R1(config-rpl-if)# else
RP/0/RP0/CPU0:R1(config-rpl-else)# pass
RP/0/RP0/CPU0:R1(config-rpl-else)# endif
RP/0/RP0/CPU0:R1(config-rpl)# end-policy
```

2. Apply to the eBGP neighbor (inbound):

```
RP/0/RP0/CPU0:R1(config)# router bgp <YOUR_ASN>
RP/0/RP0/CPU0:R1(config-bgp)# neighbor <ISP_A_PEER_IP>
RP/0/RP0/CPU0:R1(config-bgp-nbr)# address-family ipv4 unicast
RP/0/RP0/CPU0:R1(config-bgp-nbr-af)# route-policy SET_LP_ISP_A_RPL in
```

3. Set Local Preference for ALL Routes from a Neighbor:

Use a route-map/RPL that matches all routes (or has no explicit match criteria for prefixes) and applies the set local-preference action.

- **IOS XE:**

```
R1(config)# route-map SET_LP_ALL_FROM_NEIGHBOR permit 10
R1(config-route-map)# set local-preference 150
R1(config)# router bgp <YOUR_ASN>
R1(config-router)# neighbor <NEIGHBOR_IP> route-map SET_LP_ALL_FROM_NEIGHBOR in
```

- **IOS XR:**

```
RP/0/RP0/CPU0:R1(config)# route-policy SET_LP_ALL_RPL
RP/0/RP0/CPU0:R1(config-rpl)# set local-preference 150
RP/0/RP0/CPU0:R1(config-rpl)# end-policy // Implicit pass
!
RP/0/RP0/CPU0:R1(config)# router bgp <YOUR_ASN>
RP/0/RP0/CPU0:R1(config-bgp)# neighbor <NEIGHBOR_IP>
RP/0/RP0/CPU0:R1(config-bgp-nbr)# address-family ipv4 unicast
RP/0/RP0/CPU0:R1(config-bgp-nbr-af)# route-policy SET_LP_ALL_RPL in
```

Local Preference in Best Path Selection Algorithm

In the BGP best path selection process (on Cisco routers):

1. Highest **Weight** (Cisco proprietary, local to router) is preferred.
2. If Weights are equal, the path with the highest **Local Preference** is preferred.

Verifying Local Preference

- **IOS XE:** `show ip bgp`, `show ip bgp <prefix>`
- **IOS XR:** `show bgp ipv4 unicast`, `show bgp ipv4 unicast <prefix>`

Difference Between Local Preference and Weight

- **Weight:** Cisco proprietary, local to the router where configured, not advertised to any BGP peers. Influences path selection *on that router only*.

- **Local Preference:** Standard BGP attribute, advertised to all **iBGP peers** within the same AS (and confederation sub-ASes). Influences outbound path selection for the **entire AS**.

BGP AS_PATH

BGP Path Attribute: AS_PATH

The AS_PATH is a **Well-Known Mandatory** path attribute (Attribute Type Code 2). This means all BGP implementations must recognize it, and it must be included in BGP UPDATE messages.

Why is AS_PATH Needed?

The AS_PATH attribute serves two primary purposes:

1. **Loop Prevention (eBGP):** This is its most critical role. When a BGP router receives an UPDATE from an eBGP peer, it checks the AS_PATH list. If the router sees **its own AS number already present in the AS_PATH**, it means receiving this route would create a routing loop. The router will then **drop the UPDATE packet** for that prefix.

2. **Best Path Selection:** In the BGP best path selection algorithm, a **shorter AS_PATH length is preferred** over a longer one, assuming all preceding attributes (like Weight and Local Preference) are equal.

What is AS_PATH Prepending?

AS_PATH prepending is a technique used to influence **inbound traffic** by making a particular path *less attractive* to neighboring ASes.

- **How it Works:** You artificially increase the length of the AS_PATH by adding one or more instances of your own AS number (or other AS numbers, though prepending your own is the standard and safest practice) to the *beginning (left side)* of the AS_PATH when advertising routes to specific eBGP peers.

- **"Prepending":** It's called prepending because AS numbers are added to the front (leftmost side) of the AS_PATH list as routes are advertised from one AS to another. You can only prepend; you cannot remove AS numbers from the middle or end of a received AS_PATH.

- **Influence:** By making the AS_PATH appear longer via a certain peer, you make that path less desirable to that peer's AS and other downstream ASes, thus influencing them to choose an alternative path (if available) to reach your prefixes. This effectively influences how traffic *enters* your AS.

INFO

When a BGP router originates a route (e.g., via the network command), the AS_PATH is initially empty. When it advertises this route to an eBGP peer, it prepends its own AS number to the AS_PATH. Each subsequent AS that propagates the route will prepend its own AS number.

- **Reading AS_PATH:** When you view an AS_PATH (e.g., `65001 65002 65003`), you read it from left to right.
    - The **leftmost AS** (`65001`) is the AS of the eBGP neighbor that advertised the route to your router.
    - The **rightmost AS** (`65003`) is the originating AS of the prefix.

WARN

When prepending, the standard and safest practice is to prepend your own AS number multiple times. Adding AS numbers of other, unrelated ASes can cause routes to be dropped due to loop detection or lead to unexpected routing if those ASNs genuinely exist in the path.

How to Change AS_PATH (AS_PATH Prepending)

You typically use route maps (IOS XE) or route policies (IOS XR) applied **outbound** to an eBGP neighbor.

1. Prepend AS_PATH for All Routes Advertised to a Neighbor:

- Create a route map/policy that does not have a specific `match` clause (so it applies to all routes) and use the `set as-path prepend` command.
    - **IOS XE:**

```
R1(config)# route-map PREPEND_ALL permit 10
R1(config-route-map)# set as-path prepend <YOUR_AS> <YOUR_AS> // Prepend your AS twice
R1(config)# router bgp <YOUR_AS>
R1(config-router)# neighbor <NEIGHBOR_IP> route-map PREPEND_ALL out
```

```
! Or for Address Family CLI:
R1(config-router)# address-family ipv4 unicast
R1(config-router-af)# neighbor <NEIGHBOR_IP> route-map PREPEND_ALL out
```

- **IOS XR:**

```
RP/0/0/CPU0:R1(config)# route-policy PREPEND_ALL_RP
RP/0/0/CPU0:R1(config-rpl)# set as-path prepend <YOUR_AS> <YOUR_AS>
RP/0/0/CPU0:R1(config-rpl)# end-policy
!
RP/0/0/CPU0:R1(config)# router bgp <YOUR_AS>
RP/0/0/CPU0:R1(config-bgp)# neighbor <NEIGHBOR_IP>
RP/0/0/CPU0:R1(config-bgp-nbr)# address-family ipv4 unicast
RP/0/0/CPU0:R1(config-bgp-nbr-af)# route-policy PREPEND_ALL_RP out
```

2. Prepend AS_PATH for Specific Prefixes Advertised to a Neighbor:

- Match the desired prefixes using an ACL or Prefix-List within your route map/policy, then apply `set as-path prepend`.

- **IOS XE:**

```
R1(config)# ip prefix-list PL_SPECIFIC permit 10.10.0.0/16
R1(config)# route-map PREPEND_SPECIFIC permit 10
R1(config-route-map)# match ip address prefix-list PL_SPECIFIC
R1(config-route-map)# set as-path prepend <YOUR_AS> <YOUR_AS>
R1(config-route-map)# exit
R1(config)# route-map PREPEND_SPECIFIC permit 20 // Allow other routes
! Apply to neighbor outbound
```

- **IOS XR:**

```
RP/0/0/CPU0:R1(config)# prefix-set PS_SPECIFIC
RP/0/0/CPU0:R1(config-pfx)# 10.10.0.0/16
RP/0/0/CPU0:R1(config-pfx)# end-set
!
RP/0/0/CPU0:R1(config)# route-policy PREPEND_SPECIFIC_RP
RP/0/0/CPU0:R1(config-rpl)# if destination in PS_SPECIFIC then
RP/0/0/CPU0:R1(config-rpl-if)# set as-path prepend <YOUR_AS> <YOUR_AS>
RP/0/0/CPU0:R1(config-rpl-if)# else
RP/0/0/CPU0:R1(config-rpl-else)# pass
RP/0/RP0/CPU0:R1(config-rpl-else)# endif
RP/0/RP0/CPU0:R1(config-rpl)# end-policy
! Apply to neighbor outbound
```

Direction of Route Map/Policy:

- Applying **outbound** (`out`): You influence how your neighbor sees your routes (and thus how they might send traffic *to you*).

- Applying **inbound** (`in`): You could theoretically prepend to an AS_PATH as it's received, but this is highly unusual and not standard practice for influencing your own outbound path selection based on AS_PATH length (Local Preference or Weight would be used for that). AS_PATH prepending is primarily an outbound policy tool.

AS_PATH in the BGP Best Path Selection Algorithm

The AS_PATH attribute is a key tie-breaker:

1. Highest **Weight** (Cisco proprietary, local router).
2. Highest **Local Preference** (AS-wide preference for exit point).
3. Prefer locally originated routes (via `network` or `aggregate` or redistribution by this router).
4. Prefer path with the **shortest AS_PATH length**.

INFO

ISPs often use AS_PATH prepending as a tool to influence their peers because if attributes like Weight (not seen by peer) and Local Preference (not sent to eBGP peers) are not factors for the receiving AS, AS_PATH length becomes a significant decision point.

Verifying AS_PATH

- **IOS XE:** `show ip bgp`, `show ip bgp <prefix>`
- **IOS XR:** `show bgp ipv4 unicast`, `show bgp ipv4 unicast <prefix>`

BGP Origin

BGP Path Attribute: ORIGIN

The ORIGIN path attribute is a **Well-Known Mandatory** attribute (Attribute Type Code 1). This means:

- **Well-Known:** All BGP implementations must recognize it.
- **Mandatory:** It must be included in all BGP UPDATE messages that advertise prefixes.
- **Transitive:** It is passed along from one AS to another in BGP UPDATE messages.

Why Do We Need the ORIGIN Path Attribute?

The primary purpose of the ORIGIN attribute is to indicate **how a prefix was originally introduced into BGP**. This helps BGP routers determine the "believability" or preference of a route.

Most routing protocols have a way to distinguish between routes that are native to the routing domain (internally originated) versus those learned from an external source (redistributed). BGP is no different.

- **OSPF:** Uses different LSA types (e.g., Type 1/2 for internal, Type 5/7 for external/NSSA-external) and route preference logic (O > O IA > O E1/N1 > O E2/N2).

- **EIGRP:** Assigns a higher (less preferred) Administrative Distance (AD) to external routes (default AD 170) compared to internal routes (default AD 90).

- **IS-IS:** Distinguishes between internal IP reachability and external IP reachability using different TLVs or flags, and metrics/levels influence preference.

- **BGP:** Uses the ORIGIN attribute to classify routes:

    - **Origin Code `i` (IGP):** The prefix was originated into BGP within its AS of origin using the `network` **command** or by an `aggregate-address` **command**. This is the most preferred origin.

    - **Origin Code `e` (EGP):** The prefix was learned from the historic Exterior Gateway Protocol (EGP), which was BGP's predecessor. This origin code is now **obsolete** as EGP is no longer used.

    - **Origin Code `?` (Incomplete):** The origin of the prefix is unknown or learned through some other means, typically via **redistribution** from another routing protocol (like an IGP or static routes) into BGP. This is the least preferred origin.

Order of Preference for ORIGIN Codes

In the BGP best path selection algorithm, a path with a more preferred ORIGIN code will be chosen over others, assuming all preceding attributes (like Weight, Local Preference, locally originated, AS_PATH length) are equal.
The order of preference is:

1. **IGP (`i`)** - Most preferred (Numeric value 0)
2. **EGP (`e`)** - Less preferred (Numeric value 1)
3. **Incomplete (`?`)** - Least preferred (Numeric value 2)

HINT

(Since EGP (`e`) is obsolete, you'll primarily see `i` and `?` in modern networks.)

How to Change the ORIGIN Code Value

While the ORIGIN code is typically set automatically based on how a route enters BGP, you can influence it using route maps (IOS XE) or route policies (RPL on IOS XR). This is usually done to make certain paths more or less preferred.
You can change the ORIGIN attribute for:

1. **All routes** received from/sent to a neighbor.
2. **Specific routes** received from/sent to a neighbor.

- **Applying Inbound (`in`):** If you apply the policy inbound to routes received from a neighbor, you are changing how *your local router* views the origin of those routes. This affects the best-path decision on your router and consequently what your router might advertise to other iBGP peers.

- **Applying Outbound (`out`):** If you apply the policy outbound to routes being sent to a neighbor, you are changing how *that neighbor* (and subsequent ASes) will see the origin of those routes.

IOS XE (using Route-Map):

1. Match prefixes using an ACL or Prefix-List if changing for specific routes.
2. Create a Route-Map:

```
R1(config)# route-map SET_ORIGIN_RM permit 10
R1(config-route-map)# match ip address prefix-list MY_PREFIXES // Optional: for specific routes
R1(config-route-map)# set origin <igp | egp | incomplete>
R1(config-route-map)# exit
R1(config)# route-map SET_ORIGIN_RM permit 20 // Allows other routes
```

3. Apply to the BGP neighbor:

```
R1(config)# router bgp <YOUR_ASN>
! For Classic CLI:
R1(config-router)# neighbor <NEIGHBOR_IP> route-map SET_ORIGIN_RM <in | out>
! For Address Family CLI:
R1(config-router)# address-family ipv4 unicast
R1(config-router-af)# neighbor <NEIGHBOR_IP> route-map SET_ORIGIN_RM <in | out>
```

IOS XR (using Route Policy Language - RPL):

1. Create a Route Policy:

```
RP/0/RP0/CPU0:R1(config)# route-policy SET_ORIGIN_RPL
RP/0/RP0/CPU0:R1(config-rpl)# if destination in MY_PREFIX_SET then // Optional: for specific routes
RP/0/RP0/CPU0:R1(config-rpl-if)# set origin <igp | egp | incomplete>
RP/0/RP0/CPU0:R1(config-rpl-if)# else
RP/0/RP0/CPU0:R1(config-rpl-else)# pass
RP/0/RP0/CPU0:R1(config-rpl-else)# endif
RP/0/RP0/CPU0:R1(config-rpl)# end-policy
! For all routes:
! route-policy SET_ORIGIN_ALL_RPL
!   set origin <igp | egp | incomplete>
! end-policy
```

2. Apply to the BGP neighbor (under address-family):

```
RP/0/RP0/CPU0:R1(config)# router bgp <YOUR_ASN>
RP/0/RP0/CPU0:R1(config-bgp)# neighbor <NEIGHBOR_IP>
RP/0/RP0/CPU0:R1(config-bgp-nbr)# address-family ipv4 unicast
RP/0/RP0/CPU0:R1(config-bgp-nbr-af)# route-policy SET_ORIGIN_RPL <in | out>
```

Sequence of ORIGIN in BGP Best Path Selection

The ORIGIN attribute is an important tie-breaker in the BGP best path selection algorithm. On Cisco routers, it's typically considered as the **fifth step**, after:

1. Highest Weight
2. Highest Local Preference
3. Locally originated by this router (via `network`, `aggregate-address`, or self-originated `redistribute`)

4. Shortest AS_PATH length
5. **Lowest Origin code (IGP < EGP < Incomplete)**

Verifying ORIGIN Value

You can see the ORIGIN code for BGP paths using:

- **IOS XE:** `show ip bgp`, `show ip bgp <prefix>`
- **IOS XR:** `show bgp ipv4 unicast`, `show bgp ipv4 unicast <prefix>`

BGP MED

BGP Path Attribute: MED (Multi-Exit Discriminator)

The MED, also sometimes referred to simply as the "metric," is an **Optional Non-Transitive** BGP path attribute (Attribute Type Code 4).

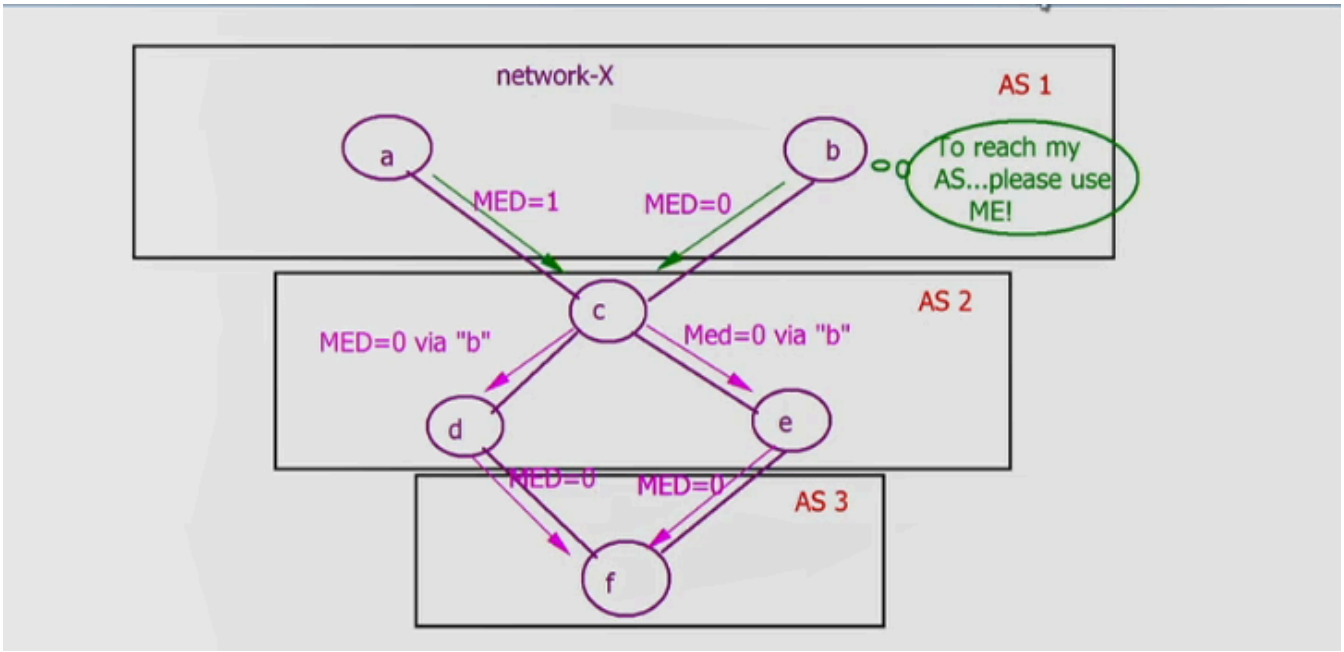
- **Optional:** BGP implementations are not strictly required to recognize or process the MED, though most major vendors do.

- **Non-Transitive:** This is a key characteristic. When an AS (AS1) sends MED values with its route advertisements to a neighboring AS (AS2), AS2 will use these MEDs for its path selection. However, AS2 **will not propagate these MED values further** to another AS (e.g., AS3). The MED's influence is typically limited to the directly peering AS.

Why Do We Need MED? Influencing Inbound Traffic

The primary purpose of MED is to allow an AS to indicate its preferred entry point(s) to a neighboring AS, thereby influencing how that neighboring AS routes traffic *towards* the AS that advertised the MED.

Case 1: Multiple Links to the Same Neighboring AS



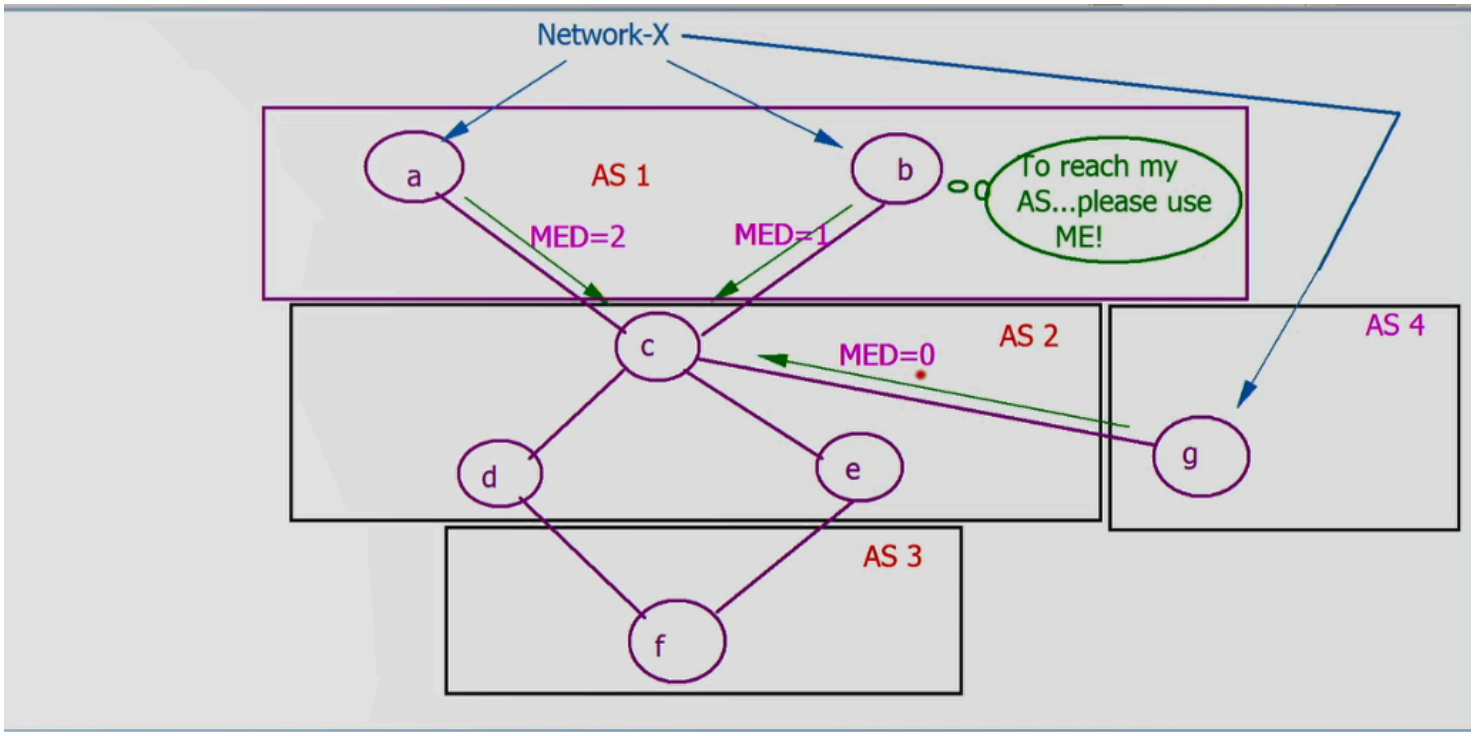
- **Scenario:** Your AS (AS1) has multiple BGP peering points (e.g., router A and router B) with a single neighboring AS (AS2, router C). You want to tell AS2 which of these links it should prefer when sending traffic to a specific prefix (Network-X) located within your AS1.

- **Solution with MED:**
  - Router A (in AS1) advertises Network-X to C (in AS2) with a MED value (e.g., MED=1).
  - Router B (in AS1) advertises Network-X to C (in AS2) with a *different* MED value (e.g., MED=0).

- **Neighbor's Decision (Router C in AS2):** Router C will compare the MED values for the paths to Network-X learned from AS1. Since a **lower MED is preferred**, Router C will choose the path via Router B (MED=0) as the best path to send traffic to Network-X in AS1.

- **Attribute Comparison Context:** In this scenario, Router C in AS2 would typically compare MED after other attributes.

Case 2: Comparing MED from Different Neighboring ASes



- **Default Behavior:** By default, BGP routers **only compare MED values for paths received from the same neighboring AS**. So, in the diagram, Router C would compare the MED from A (AS1) and B (AS1) and prefer B. It would *not* by default compare the MED from B (AS1) with the MED from G (AS4).

- **Changing Default Behavior:** To force BGP to compare MED values for paths originating from *different* neighboring ASes (assuming preceding path attributes are equal), you can use specific commands:
  - **IOS XE:** `R1(config-router)# bgp always-compare-med`

- **IOS XR:** `RP/0/0/CPU0:R1(config-bgp)# bgp bestpath med always`

MED Values

- **Range:** MED is a 32-bit unsigned integer, so values can range from 0 to 4,294,967,295.
- **Preference: Lower MED values are more preferred.**
- **Default Value:**
  - If an UPDATE message is received without a MED attribute, Cisco IOS routers historically treated this as a MED of 0 (most preferred). This behavior can be changed with the `bgp bestpath med missing-as-worst` command, which makes paths without a MED the least preferred (as if they had the highest MED).
  - When routes are redistributed from an IGP into BGP on a Cisco router, the IGP metric is often copied into the MED by default. If no metric is available from the IGP or if a default is needed, the `default-metric <value>` command under `router bgp` can set a MED for redistributed routes.

How to Change/Set MED Values

MED is typically set using route maps (IOS XE) or route policies (RPL on IOS XR) applied **outbound** to an eBGP neighbor. This allows your AS to tell the neighboring AS which path is preferred for traffic coming *into* your AS.

1. Set MED for All Routes Advertised to a Neighbor:

- **IOS XE:**

```
R1(config)# route-map SET_MED_OUT permit 10
R1(config-route-map)# set metric 50 // Sets MED to 50
R1(config)# router bgp <YOUR_ASN>
R1(config-router)# neighbor <NEIGHBOR_IP> route-map SET_MED_OUT out
```

- **IOS XR:**

```
RP/0/RP0/CPU0:R1(config)# route-policy SET_MED_OUT_RPL
RP/0/RP0/CPU0:R1(config-rpl)# set med 50
```

```
RP/0/RP0/CPU0:R1(config-rpl)# end-policy
!
RP/0/RP0/CPU0:R1(config)# router bgp <YOUR_ASN>
RP/0/RP0/CPU0:R1(config-bgp)# neighbor <NEIGHBOR_IP>
RP/0/RP0/CPU0:R1(config-bgp-nbr)# address-family ipv4 unicast
RP/0/RP0/CPU0:R1(config-bgp-nbr-af)# route-policy SET_MED_OUT_RPL out
```

2. Set MED for Specific Prefixes Advertised to a Neighbor:

Combine matching (ACL, prefix-list, or prefix-set in RPL) with the set metric action within your route map/policy.

3. Set Default MED for Redistributed Routes:

- **IOS XE & IOS XR:**

```
R1(config)# router bgp <YOUR_ASN>
R1(config-router)# default-metric <MED_VALUE> // For IOS XE
!
RP/0/RP0/CPU0:R1(config)# router bgp <YOUR_ASN>
RP/0/RP0/CPU0:R1(config-bgp)# default-metric <MED_VALUE> // For IOS XR
```

This applies when redistributing routes into BGP if their original metric isn't carried into MED.

Sequence of MED in BGP Best Path Selection

MED is considered after several other attributes. In Cisco's BGP best path algorithm, it's typically step #6:

1. Highest Weight
2. Highest Local Preference
3. Locally originated by this router
4. Shortest AS_PATH length
5. Lowest Origin type (IGP < EGP < Incomplete)
6. **Lowest MED** (Compared only for paths from the same neighboring AS, unless `bgp always-compare-med` or `bgp bestpath med always` is configured).

Treating Missing MED as Worst-Case

As noted, if a path is received without a MED, it's often treated as MED 0. To change this:

- **IOS XE:** `R1(config-router)# bgp bestpath med missing-as-worst`
- **IOS XR:** `RP/0/0/CPU0:R1(config-bgp)# bgp bestpath med missing-as-worst`

Verifying MED Value

- **IOS XE:** `show ip bgp`, `show ip bgp <prefix>`
- **IOS XR:** `show bgp ipv4 unicast`, `show bgp ipv4 unicast <prefix>`

BGP Communities

BGP Path Attribute: Communities

The BGP Community attribute (Attribute Type Code 8) is an **Optional Transitive** attribute.

- **Optional:** BGP implementations are not required to support it, though virtually all modern routers do.

- **Transitive:** If a BGP speaker receives a route with a community it doesn't recognize or act upon, it should pass the community along to its other BGP peers (both iBGP and eBGP, if configured to send communities).

Why Do We Need BGP Communities?

While the Community attribute itself is **not directly used as a step in the BGP Best Path Selection Algorithm** (like MED or Local Preference are), it is extremely useful for:

1. **Route Tagging:** Assigning one or more "tags" (community values) to a set of routes.

2. **Policy Implementation:** Routers can then be configured with policies (using route maps or RPL) to match on these community values and take specific actions, such as:

    - **Route Filtering:** Permitting or denying routes based on their community tags.
    - **Influencing Path Selection:** Modifying other BGP attributes (like Local Preference, MED, or prepending AS_PATH) for routes tagged with specific communities.
    - **Controlling Route Propagation:** Using well-known communities to limit how far a route is advertised.

Essentially, communities allow for grouping routes that share a common policy characteristic, simplifying configuration and management.

Community Values and Format

- A BGP Community is a 32-bit number.

- It's typically displayed and configured in an `AA:NN` format, where `AA` is usually an AS number and `NN` is a locally significant number within that AS. This helps ensure global uniqueness and provides context.

- Example: `65001:100`

- **Display Format Command:**

  - **IOS XE:** By default, IOS might display the community as a single decimal number. To display it in the more readable `AA:NN` format:

    ```
    R1(config)# ip bgp-community new-format
    ```

  - **IOS XR & NX-OS:** Typically display communities in the `AA:NN` format by default.

Sending Communities to Neighbors

How communities are sent to BGP peers differs slightly by platform and peer type:

- **IOS XE:**

  - **To iBGP Peers:** Standard communities are generally **sent by default** without explicit configuration.

  - **To eBGP Peers:** Standard communities are **NOT sent by default**. You must explicitly enable sending them.

    ```
    R1(config)# router bgp YOUR_ASN
    ! For Classic CLI:
    R1(config-router)# neighbor NEIGHBOR_IP send-community
    ! To send both standard and extended communities (for MPLS VPNs etc.):
    R1(config-router)# neighbor NEIGHBOR_IP send-community both

    ! For Address Family CLI:
    R1(config-router)# address-family ipv4 unicast
    R1(config-router-af)# neighbor NEIGHBOR_IP send-community [standard | extended | both]
    ```

- **IOS XR:**

- **To iBGP Peers:** Standard and extended communities are generally **sent by default**.

- **To eBGP Peers:** Communities are **NOT sent by default**.

```
RP/0/RP0/CPU0:R1(config-bgp-nbr-af)# send-community-ebgp // Sends standard communities to t
RP/0/RP0/CPU0:R1(config-bgp-nbr-af)# send-extended-community-ebgp // Sends extended communi
! To send both, configure both commands.
```

Well-Known BGP Communities

These are specific community values with pre-defined meanings and actions that most BGP implementations (including Cisco IOS) understand and act upon automatically:

1. `NO_ADVERTISE` **(Value: 0xFFFFFF02 or** `no-advertise` **)**
   - **Action:** When a BGP router receives a route tagged with this community, it **will use the route locally but will NOT advertise it to ANY other BGP peer** (neither iBGP nor eBGP).

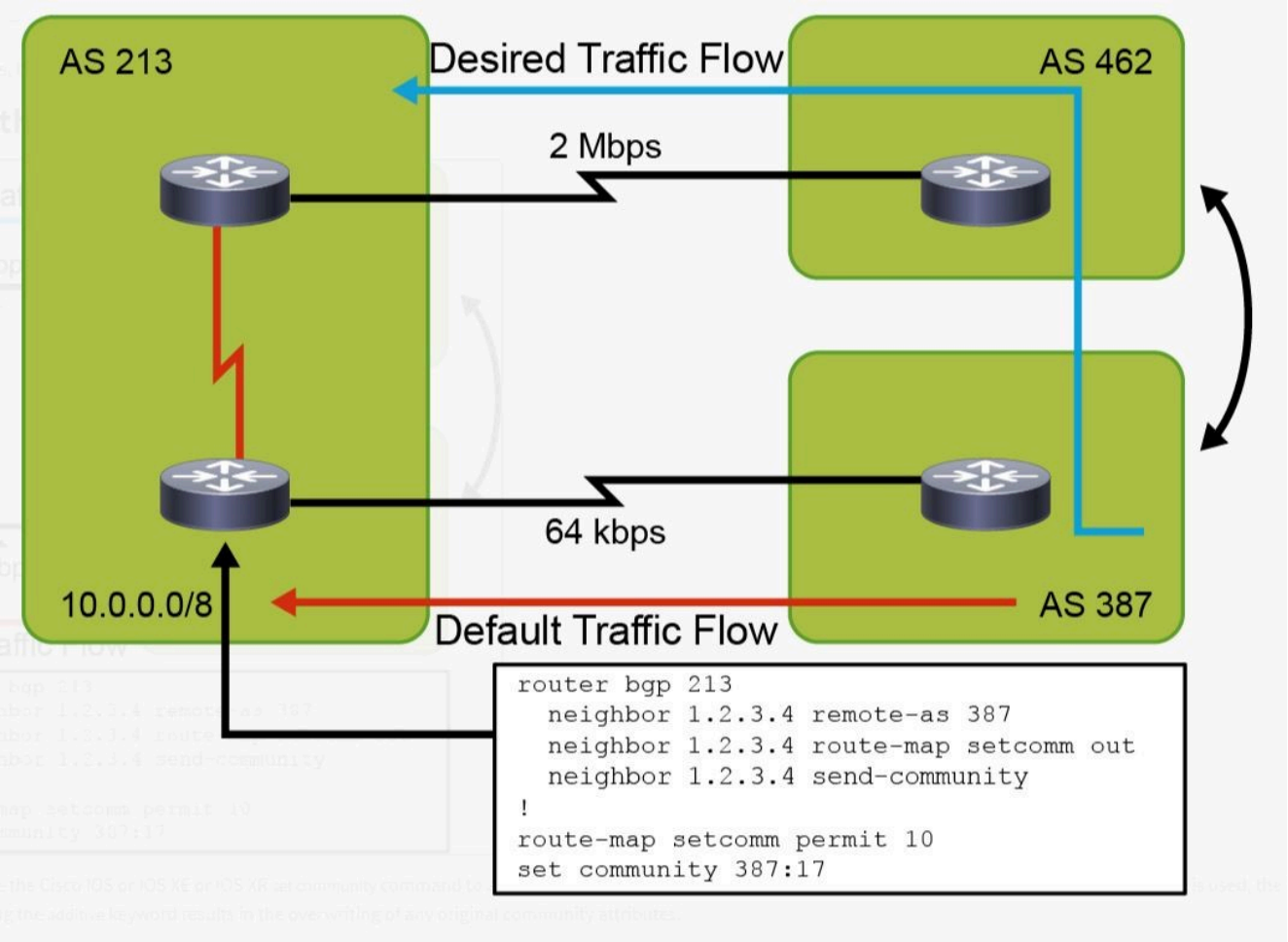
2. `NO_EXPORT` **(Value: 0xFFFFFF01 or** `no-export` **)**
   - **Action:** When a BGP router receives a route tagged with this community, it can advertise it to its **iBGP peers** (within the same AS, or to other member-ASes within the same BGP confederation). However, it **will NOT advertise this route to any eBGP peers** outside its local AS (or outside the confederation boundary).

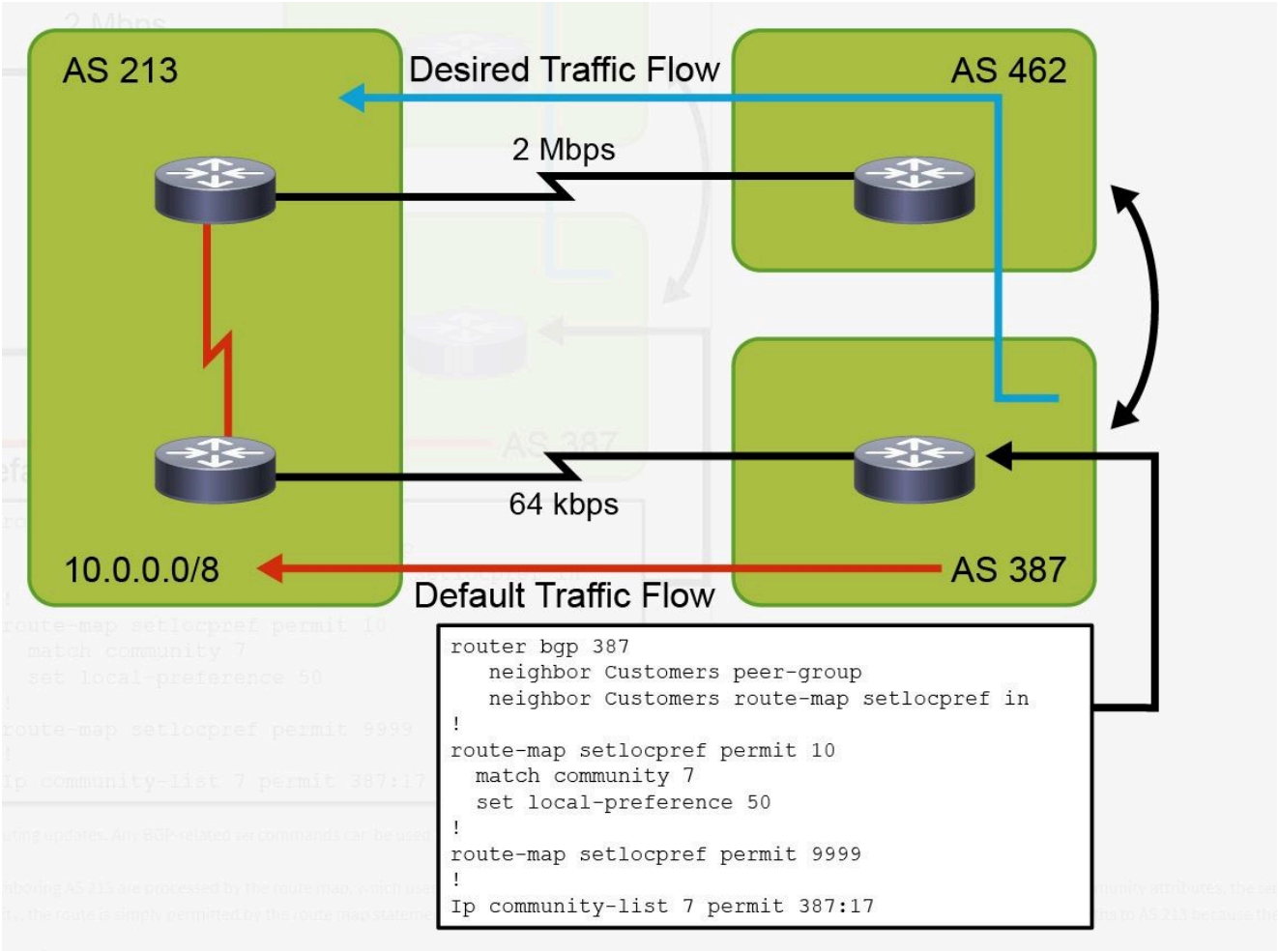
3. `LOCAL_AS` **(Value: 0xFFFFFF03 - Cisco name) /** `NO_EXPORT_SUBCONFED` **(RFC name)**
   - **Action:** This community is primarily relevant within BGP Confederations. When a route is tagged with this community:
     - It can be advertised to iBGP peers **within the same member-AS** of the confederation.
     - It will **NOT be advertised to eBGP peers in other member-ASes** within the same confederation.
     - It will also **NOT be advertised to any eBGP peers outside the overall confederation**.
   - If not using confederations, its behavior is very similar to `NO_EXPORT`.

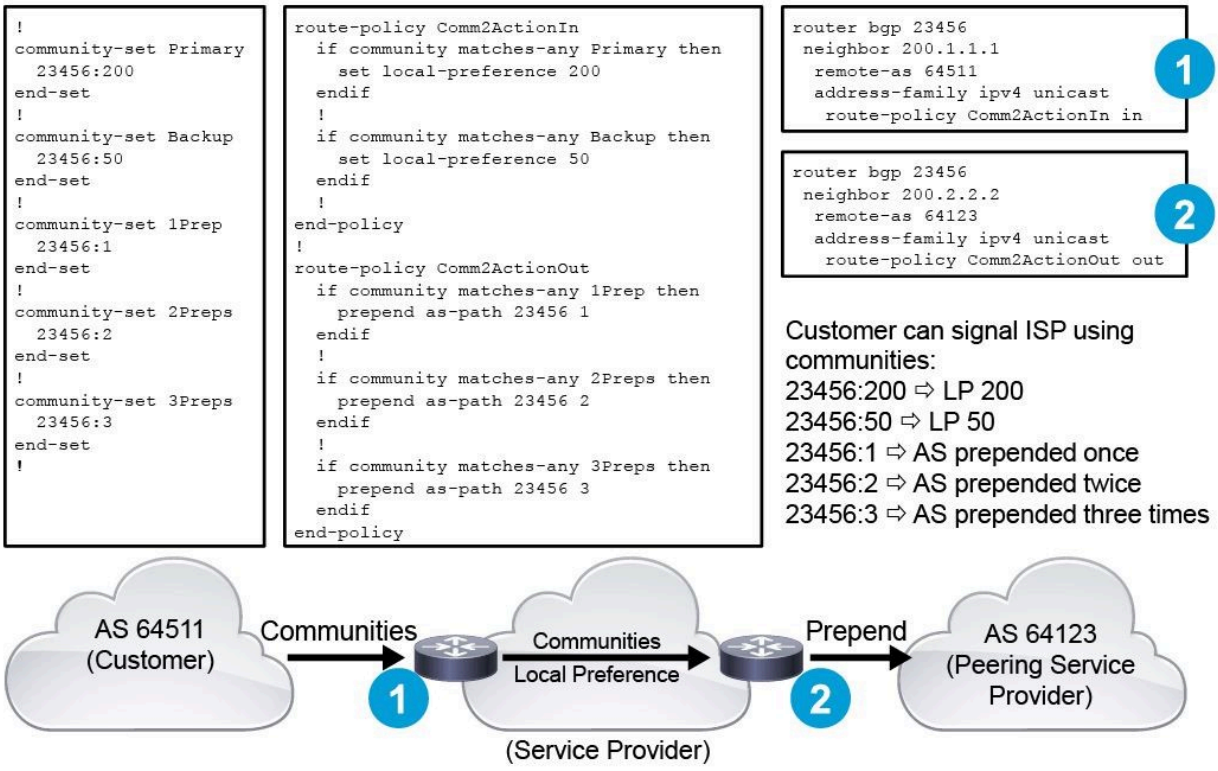
Setting and Matching Community Values

Communities are typically set and matched using route maps (IOS XE) or Route Policy Language (RPL on IOS XR).

- **Setting Communities:**
  - The `set community` command is used.
  - `set community ;community_value; ;community_value_2; ... [additive]`
  - If the `additive` keyword is used, the specified communities are added to any existing communities on the route.
  - If `additive` is omitted, the specified communities **overwrite** any existing communities.
  - You can also use `set community none` to remove all communities.







- **Matching Communities:**

- In route maps/RPL, you can match routes based on the community values they carry using `match community ;community-list_name`.
- Community-lists (similar to ACLs but for community values) are used to define specific communities or patterns to match.

BGP Best Path Selection Algorithm

BGP Best Path Selection Algorithm & Multipath

When a BGP router learns multiple paths to the same destination prefix, it must decide which single path (or paths, if multipath is enabled) is the "best." This decision is made using a systematic, step-by-step process known as the BGP Best Path Selection Algorithm, which evaluates various BGP Path Attributes.

The Purpose of Path Attributes & Best Path Selection

- **Path Attributes Reviewed:** As we've discussed, BGP UPDATE messages carry path attributes like NEXT_HOP, AS_PATH, ORIGIN, LOCAL_PREFERENCE, MED, etc. These attributes describe the characteristics of a route.
    - **Well-Known Attributes** (e.g., AS_PATH, NEXT_HOP, ORIGIN, LOCAL_PREFERENCE) must be recognized by all BGP implementations. Some are mandatory, others discretionary.
    - **Optional Attributes** (e.g., MED, COMMUNITY) provide additional policy flexibility.
    - It's important to remember that an attribute like **Local Preference**, while well-known and transitive within an AS (and confederations), is not advertised to external eBGP peers in different ASes.

The BGP Best Path Selection Algorithm Steps (Cisco)

BGP routers compare paths to the same prefix by going through a list of criteria. As soon as a path is found to be superior based on one attribute, it is selected as the best path, and further criteria are typically not evaluated for that prefix .

Prerequisite: Before a path even enters the selection process, the **BGP NEXT_HOP attribute for that path must be reachable** in the router's RIB (Routing Information Base) via a non-BGP route (IGP, static, or connected). If the next-hop is unreachable, the path is considered invalid and not a candidate for best path.
Here's a commonly referenced order of the BGP best path selection steps on Cisco routers:

1. **Highest WEIGHT:** Prefer the path with the highest Weight. (Cisco proprietary, local to the router).
2. **Highest LOCAL_PREFERENCE:** Prefer the path with the highest Local Preference. (Used within an AS to choose a consistent exit point).
3. **Locally Originated:** Prefer paths that were originated locally by this router (via `network`, `aggregate-address`, or redistribution) over BGP Learned ones.
4. **Shortest AS_PATH Length:** Prefer the path with the shortest AS_PATH length. (Fewer AS hops is generally better).
5. **Lowest ORIGIN Code:** Prefer the path with the lowest origin type:
    - `i` (IGP) is better than
    - `e` (EGP - obsolete) which is better than
    - `?` (Incomplete - typically redistributed routes).

6. **Lowest MED (Multi-Exit Discriminator):** Prefer the path with the lowest MED.
    - By default, MED is only compared among paths learned from the **same neighboring AS**.
    - The `bgp always-compare-med` (IOS/XE) or `bgp bestpath med always` (XR) command allows MED comparison across different neighboring ASes.
    - The `bgp bestpath med missing-as-worst` command treats paths without a MED as having the highest (worst) MED.

7. **eBGP over iBGP:** Prefer paths learned from an eBGP peer over paths learned from an iBGP peer.

8. **Lowest IGP Metric to BGP Next-Hop:** Prefer the path with the lowest IGP metric to reach the BGP next-hop address. (This helps choose the closest exit/entry point within the local AS if multiple iBGP paths exist or if eBGP next-hops are internal).

9. **Multipath Consideration (if enabled):** If multiple paths are equal up to this point (and meet multipath criteria, see below), BGP may select multiple paths for load sharing. If multipath is not enabled or paths are not eligible, proceed to further tie-breakers.

10. **Oldest Path (for eBGP paths):** When comparing multiple eBGP paths that are otherwise equal, prefer the path that was learned first (the oldest one). This helps minimize route flapping.

11. **Lowest Router ID of Peer:** Prefer the path from the BGP peer with the lowest BGP Router ID.

12. **Shortest CLUSTER_LIST Length:** If paths are learned via Route Reflectors, prefer the path with the shortest CLUSTER_LIST length.

13. **Lowest Neighbor IP Address:** Prefer the path from the BGP peer with the lowest neighbor IP address.

HINT

(Mnemonics like "We Love Oranges As Oranges Mean Pure Refreshment" are sometimes used to remember the order of key attributes: Next-Hop (validity), Weight, Local_Pref, AS_PATH, Origin, MED, eBGP over iBGP, IGP metric, Router ID.)

BGP Multipath: Using Multiple "Best" Paths

By default, BGP selects only one single best path per prefix to install in the Routing Information Base (RIB) and advertise to peers. However, the **BGP Multipath feature** allows a router to install multiple BGP paths for the same destination prefix into the RIB. This enables load balancing or load sharing across these multiple paths.

Conditions for Paths to be Eligible for Multipath:

For BGP to consider multiple paths for multipathing, those paths must be "equally preferable" based on the evaluation of BGP path attributes up to a certain point in the best path selection algorithm. The specific attributes that must match can vary slightly based on whether the paths are eBGP-learned or iBGP-learned, and some platform-specific behaviors or additional configuration knobs (like `bgp bestpath as-path multipath-relax`) can influence this.

Generally, for paths to be considered for multipath:

- **For paths learned from eBGP peers:**
    - They must typically have the same **Weight**.
    - They must have the same **Local Preference**.
    - Neither path should be locally originated by the router itself.

    - They must have the same **AS_PATH length**.
    - They must have the same **Origin code** (IGP, EGP, or Incomplete).
    - They must have the same **MED**
    - The BGP next-hop addresses for each path must be different and resolvable.

- **For paths learned from iBGP peers**
    - They must typically have the same **Weight**.
    - They must have the same **Local Preference** (this is key for iBGP).
    - Neither path should be locally originated by the local router.
    - They must have the same **AS_PATH**
    - They must have the same **Origin code**.
    - They must have the same **MED**
    - Crucially for iBGP multipath, they must have the **same IGP metric to reach their respective BGP next-hop addresses.**
    - The BGP next-hop addresses must be different and resolvable.

Behavior with Multipath Enabled:

- If multiple paths satisfy the equality criteria for multipath (i.e., they are identical for all attributes checked *before* the later tie-breaking steps like oldest path or router ID), BGP will select these paths as candidates.

- The router will then install up to the configured `maximum-paths` number of these equally preferable paths into the RIB.

- Importantly, when paths are selected for multipath, the BGP best path algorithm **does not proceed to the subsequent tie-breaking criteria** (such as preferring the oldest eBGP path, lowest peer Router ID, or lowest peer IP address) for those paths. Those tie-breakers are only used if a single best path still needs to be chosen from paths that were otherwise equal but multipath was not enabled or not all multipath conditions were met.

Enabling BGP Multipath (Example - IOS XE):

```
R1(config)# router bgp <YOUR_ASN>
R1(config-router)# maximum-paths <number_of_paths>        // Enables multipath for eBGP learned paths
R1(config-router)# maximum-paths ibgp <number_of_paths>   // Enables multipath for iBGP learned paths
R1(config-router)# maximum-paths eibgp <number_of_paths>  // Enables multipath for both eBGP and iBGP paths (platform dependent)
```

HINT

(IOS XR configuration is similar, with `maximum-paths ebgp <N>`, `maximum-paths ibgp <N>`, `maximum-paths eibgp <N>` configured under the BGP process or specific address family.)

BGP Route Refresh

Hard Reset vs. Soft Reset vs. Route Refresh

When you modify BGP routing policies such as changing filter lists, route maps for path attribute manipulation, or other policy configurations, these new policies need to be applied to the BGP routes. This means the BGP process must re-evaluate routes against the new policy.

---

1. Hard Reset (The "Bad" Solution)

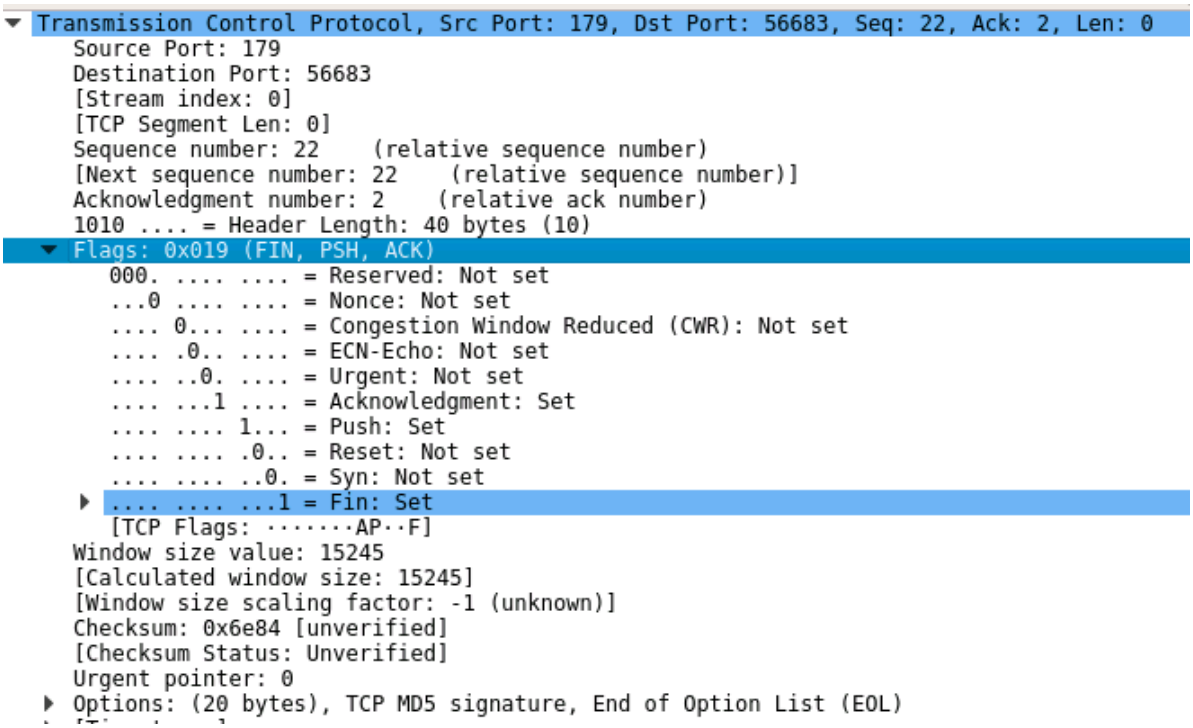
This involves completely tearing down and re-establishing the BGP peering session.

- **Commands:**
  - `clear ip bgp *` (IOS XE: Resets ALL BGP sessions)
  - `clear ip bgp <NEIGHBOR_IP>` (IOS XE: Resets session with a specific neighbor)
  - `clear bgp *` (IOS XR: Resets ALL BGP sessions)
  - `clear bgp <NEIGHBOR_IP>` (IOS XR: Resets session with a specific neighbor)

- **Why it's "Bad" to do hard reset?**

  1. **Session Teardown:** Sends a TCP FIN/RST, killing the TCP session and BGP peering.
  2. **Route Removal:** All routes learned from that peer are removed from the BGP table and consequently from the IP routing table.
  3. **Full Re-establishment:** The entire BGP FSM process starts again (TCP 3-way handshake, OPEN message exchange, etc.).
  4. **Full Re-advertisement:** All routes need to be re-exchanged and re-processed against policies.

- **Impact:** This causes a **BGP downtime** for that peering, potentially lasting several minutes in large networks. This can disrupt traffic and is generally unacceptable in production environments for simple policy changes.

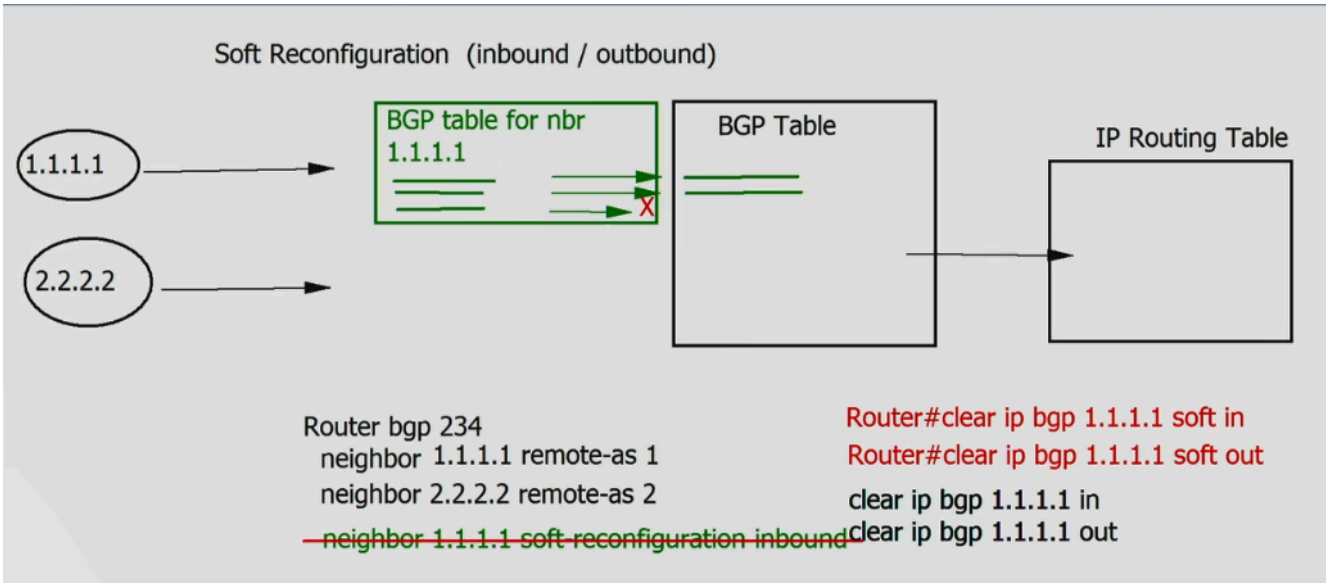


---

2. Soft Reconfiguration ("Better" Solution)

Soft reconfiguration allows BGP to re-process routes against new inbound policies without tearing down the BGP session.

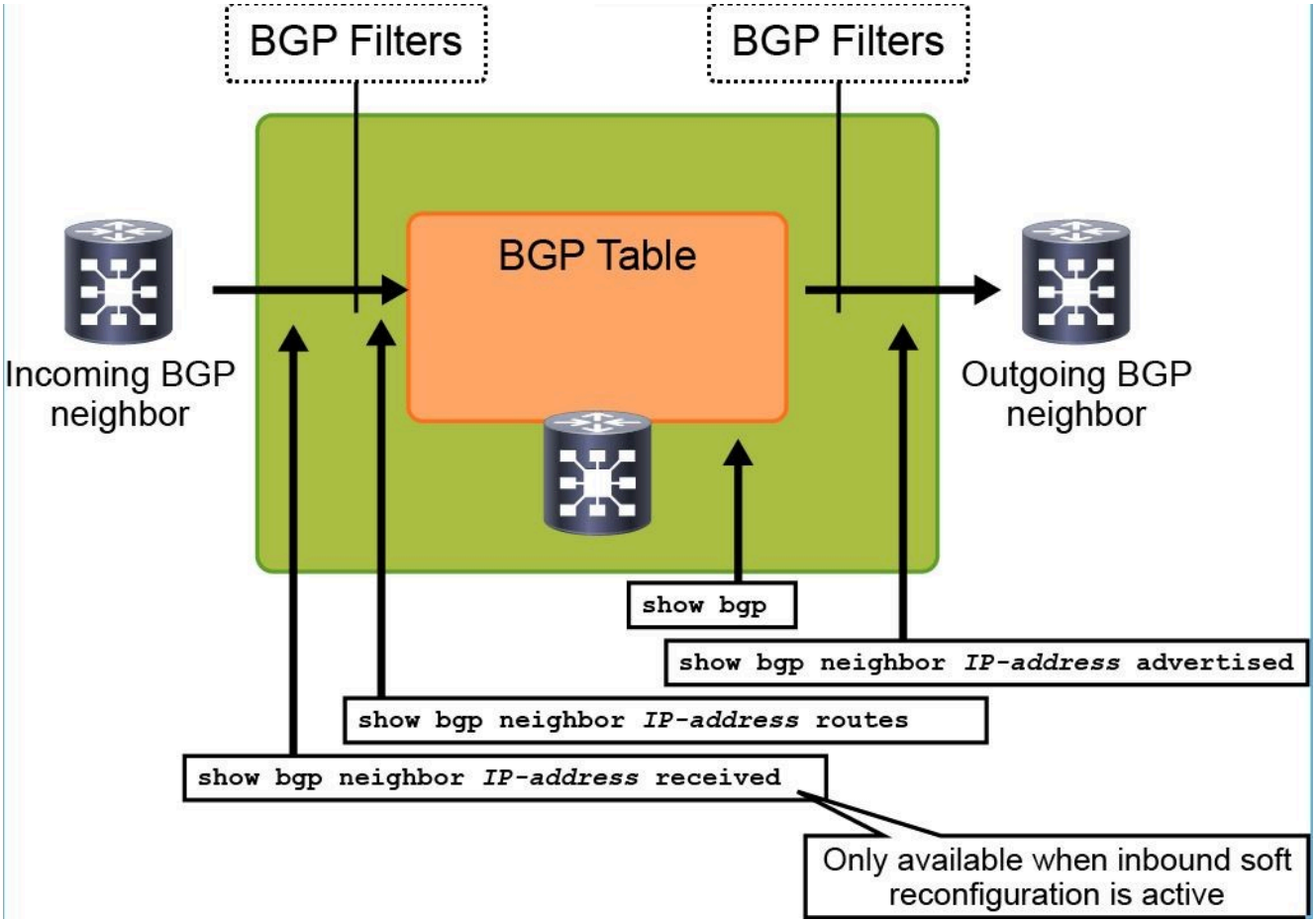
- **Mechanism (Inbound):**
  - You first configure `neighbor <NEIGHBOR_IP> soft-reconfiguration inbound` (IOS XE) on a per-neighbor basis.
  - When enabled, the router stores an **unmodified (pre-policy) copy of all routes received from that specific neighbor** in a separate area of memory. This is often referred to as the "Adj-RIB-In_unfiltered."



- **Applying New Inbound Policy:**
  - After changing your inbound route map or filter list, you use the command:
    - **IOS XE:** `clear ip bgp <NEIGHBOR_IP | *> soft in`
  - The router then takes the stored, unmodified routes from the special memory table and re-runs them through the *new* inbound policy. The resulting routes are then processed by BGP (best path, RIB installation).

- **Pros:**
  - Does not reset the BGP session; peering stays UP.
  - Avoids BGP downtime.

- **Cons:**
  - **High Memory Consumption:** Storing a full copy of all received routes from potentially many peers (especially if they send full internet tables) can consume significant router memory. This is its major drawback and why it's less favored now.



- **Viewing Stored Routes (IOS XE):**
  - `show ip bgp neighbors <NEIGHBOR_IP> received-routes` (shows the unfiltered routes stored due to `soft-reconfiguration inbound`)

- **Outbound Soft Reconfiguration:**
  - For outbound policy changes, you don't need to configure `soft-reconfiguration inbound` for the peer.

  - You simply apply the new outbound policy and then use:
    - **IOS XE:** `clear ip bgp <NEIGHBOR_IP | *> soft out`

  - This command tells the router to re-evaluate its BGP table routes against the new outbound policy for that neighbor and send out new UPDATE messages for any changed routes, without tearing down the session. It doesn't require extra memory storage for outbound.

---

3. Route Refresh Capability ("Best" Solution)

This is the modern and most efficient way to apply BGP policy changes without resetting sessions or consuming extra memory for inbound policies.

- **Mechanism:**
    - Route Refresh (RFC 2918) is a BGP capability negotiated between peers in their OPEN messages. Most modern BGP implementations support it by default.
    - If both peers support it, one router can request the other to re-send its routing information for a specific address family.
- **Applying New Policy with Route Refresh:**
    - **Inbound Policy Change (You want to re-process routes *received* from your neighbor):**
        - **IOS XE:** `clear ip bgp <NEIGHBOR_IP | *> in`
        - **IOS XR:** `clear bgp [ipv4 unicast] <NEIGHBOR_IP | *> [soft] in`

          HINT

            - *(In XR, `soft in` often defaults to route refresh if supported and soft-reconfig inbound is not explicitly configured with `always`)*
        - This sends a ROUTE-REFRESH *request* message (BGP Message Type 5) to the neighbor. The neighbor then re-advertises all its routes for the specified address family. Your router receives these routes anew and processes them against your *new* inbound policy.
    - **Outbound Policy Change (You want to re-send routes *to* your neighbor with new policy):**
        - **IOS XE:** `clear ip bgp <NEIGHBOR_IP | *> out`
        - **IOS XR:** `clear bgp [ipv4 unicast] <NEIGHBOR_IP | *> [soft] out`
        - This causes your router to re-evaluate its routes against the new outbound policy for that neighbor and send appropriate BGP UPDATE messages.
- **Pros:**
    - No session reset, no BGP downtime.

    - No extra memory consumption for storing unfiltered routes (unlike `soft-reconfiguration inbound`).
    - Standardized mechanism.

- **Special Case for `clear ip bgp <NEIGHBOR_IP> soft in` (IOS XE):**
    - If `neighbor <NEIGHBOR_IP> soft-reconfiguration inbound` **is configured:** This command uses the stored unfiltered routes.
    - If `soft-reconfiguration inbound` **is NOT configured** (and the peer supports Route Refresh): This command will automatically attempt to use the Route Refresh capability by sending a request to the peer.

IOS XR `soft-reconfiguration inbound`:

- In IOS XR, the command is `soft-reconfiguration inbound [always]`.
- If `always` is not specified, IOS XR will prefer to use Route Refresh if the peer supports it, even if `soft-reconfiguration inbound` is configured. The `always` keyword forces the use of memory-based soft reconfiguration.

Summary of BGP Clearing Commands

- **Hard Reset (`clear ip bgp <peer>` or `clear bgp <peer>`):** Tears down TCP and BGP. Avoid for policy changes if possible.

- **Soft Reconfiguration Inbound (using stored routes on IOS XE):** `neighbor <X> soft-reconfiguration inbound` (to enable storage) then `clear ip bgp <X> soft in`. Memory intensive.

- **Route Refresh (Preferred for inbound policy re-evaluation):** `clear ip bgp <X> in` (XE) or `clear bgp <X> soft in` (XR, often defaults to Route Refresh). Sends a request to peer.

- **Outbound Policy Re-evaluation:** `clear ip bgp <X> out` (XE) or `clear bgp <X> soft out` (XR). Re-sends your routes after local policy re-eval.

BGP AS_SEQ vs AS_SET

AS_PATH (AS_SEQUENCE & AS_SET)

The **AS_PATH** is a fundamental and **Well-Known Mandatory** BGP path attribute (Attribute Type Code 2). It's included in BGP UPDATE messages and describes the sequence of Autonomous Systems (ASes) a route has traversed to reach the advertising router. You can clearly see it in the output of commands like `show ip bgp`.

Why Do We Need AS_PATH?

The AS_PATH attribute serves two critical functions:

1. **Loop Prevention (Primary Role for eBGP):** This is its most important function between different Autonomous Systems. When a BGP router receives an UPDATE from an eBGP peer, it inspects the AS_PATH list. If the router sees **its own AS number already present in the AS_PATH**, it means accepting this route would create a routing loop. The router will then **discard the UPDATE** for that prefix.

2. **Best Path Selection:** The AS_PATH is also a key factor in the BGP best path selection algorithm. All else being equal, BGP prefers the path with the **shortest AS_PATH length** (i.e., the path that has traversed the fewest ASes).

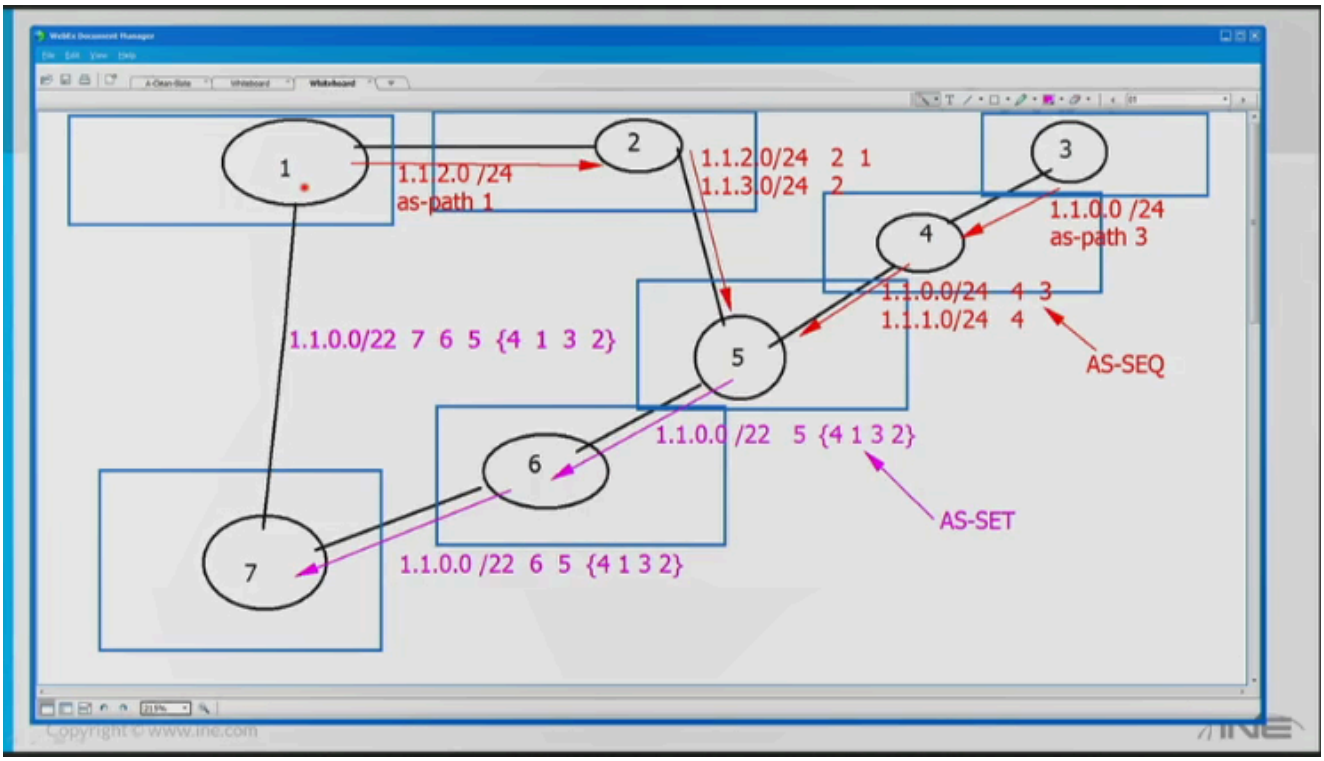
AS_SEQUENCE and AS_SET

The AS_PATH attribute is composed of segments, and each segment is of a particular type. The two main types you'll encounter are:

1. **AS_SEQUENCE (AS_SEQ):**

    - **What it is:** This is the most common type of segment. It's an **ordered set of AS numbers** that the route has traversed. Each AS that forwards the route to an eBGP peer prepends its own AS number to the AS_SEQUENCE.

    - **Example:** If a route originates in AS3, passes through AS2, and is then advertised to AS1, the AS_PATH received by AS1 would look like `2 3` (AS2 pretended its number when sending to AS1, and AS3 was the origin).



- **in the above diagram,** When Router 5 (in AS 5) receives updates from Router 2 (in AS 2) for prefix `1.1.2.0/24` (originated in AS 1), the AS_PATH would be `2 1`. Similarly, from Router 4 (in AS 4) for prefix `1.1.1.0/24` (originated in AS 3), the AS_PATH would be `4 3`. These are examples of AS_SEQUENCE.

2. **AS_SET:**

- **What it is:** This is an **unordered set of AS numbers**. It is primarily generated when **route aggregation (summarization) occurs** in BGP, and the aggregate route combines prefixes that have traversed different AS paths. To preserve AS_PATH information from the aggregated routes for loop detection without implying a specific order, the AS numbers from the paths of the aggregated routes are included in an AS_SET.

- **Representation:** Usually denoted by curly braces `{}` in `show ip bgp` outputs, e.g., `{2,4,1,3}`.

- **Example (from above diagram):** Router 5 (in AS 5) receives routes for `1.1.2.0/24` (path `2 1`) and `1.1.1.0/24` (path `4 3`). If R5 then creates an aggregate route like `1.1.0.0/22` that summarizes both of these, and it's configured to include AS_PATH information from the specifics, it might generate an AS_PATH for `1.1.0.0/22` that includes an AS_SET like `{1,2,3,4}` when advertising to R6.

HINT

When a router creates an aggregate and forms an AS_SET, the AS_SET contains AS numbers from the AS_PATHs of the routes being aggregated. The aggregating router's own AS is then prepended to this AS_SET (as an AS_SEQUENCE) when the aggregate is advertised to an eBGP peer.

AS_SET and Loop Prevention

Even if an AS number appears within an AS_SET, the BGP loop detection mechanism still applies. If a router receives an update (e.g., for an aggregate route) and sees its own AS number within an AS_SET in the AS_PATH, it will still consider this a loop and drop the update.

Why is AS_SET Needed with Aggregation?

when routes are aggregated, the original AS_PATH information of the more specific "child" routes is effectively lost or hidden by the summary. This loss of path information could break the BGP loop prevention mechanism if not handled correctly.

To address this, BGP offers two main options when creating an aggregate using the aggregate-address command:

1. **Include AS_SET (Default behavior if not suppressed and if specific routes have diverse AS_PATHs):**

    - The `aggregate-address <prefix> <mask>` command (without `summary-only` and without options to suppress AS_SET formation, and if the contributing routes have different AS_PATH information) will typically create an AS_PATH for the aggregate that includes an AS_SET. This AS_SET lists the AS numbers from the AS_PATHs of the routes being aggregated.

    - This preserves the AS path information for loop detection purposes. The router originating the aggregate will still prepend its own AS to this (potentially including the AS_SET) when sending to eBGP peers.

2. **Omit AS_SET (Using `summary-only`):**

    - If you use the `summary-only` keyword with `aggregate-address`, only the aggregate is advertised, and the more specific routes (and their AS_PATHs) are suppressed. In this case, the aggregate might not have an AS_SET if the router is just originating it as if it's new.

- **Atomic Aggregate and Aggregator Attributes:** If AS_PATH information is lost or significantly altered due to aggregation (especially if AS_SET isn't fully representative or is suppressed), BGP uses two other attributes to indicate that aggregation has occurred and information might have been lost:

    - **ATOMIC_AGGREGATE:** A Well-Known Discretionary attribute that signals that the path information for this aggregate may not be complete (i.e., a more specific route from an AS not listed in the AS_PATH may exist).

    - **AGGREGATOR:** An Optional Transitive attribute that identifies the AS number and BGP router ID of the router that performed the aggregation. These attributes help in understanding the nature of the aggregate and aid in loop prevention when full AS_PATH information from specific routes is not present in the aggregate's AS_PATH.

BGP Routes Aggregation

BGP Route Summarization (Aggregation)

BGP allows you to create a summary (aggregate) route from more specific prefixes that already exist in your BGP table.

How to Configure (`aggregate-address`)

- **IOS XE (Classic & Address Family CLI):**

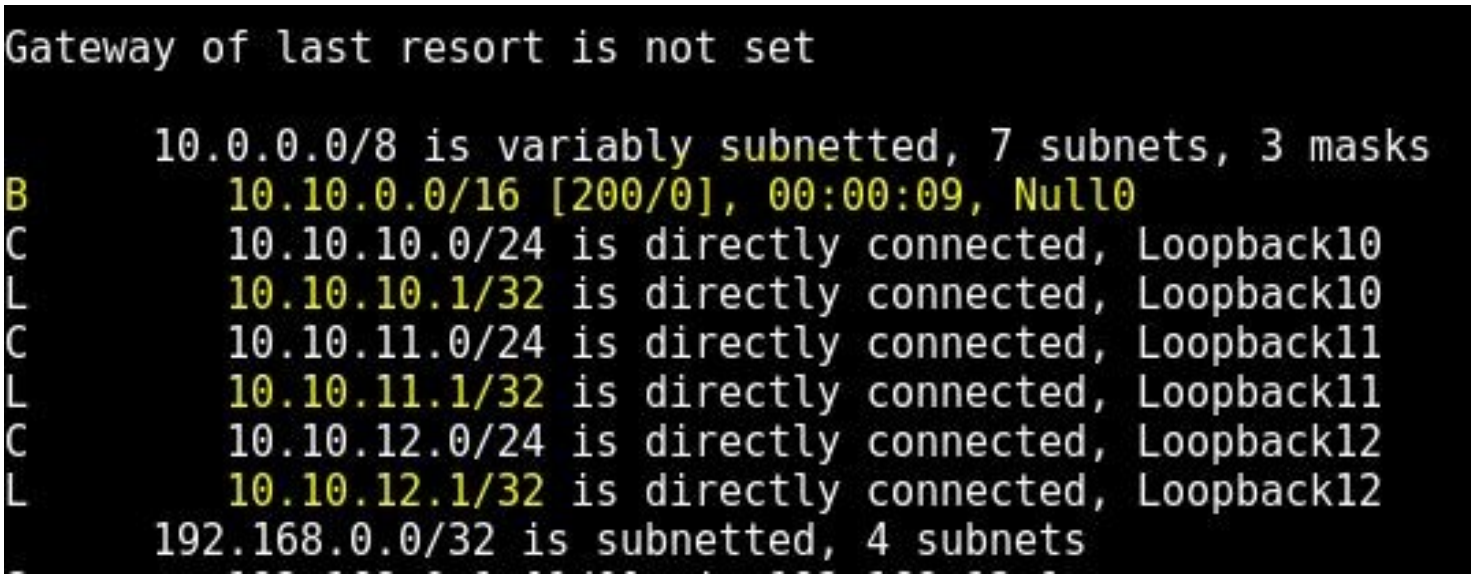
```
! Classic CLI
R1(config-router)# aggregate-address <PREFIX> <MASK> [options]
! Address Family CLI
R1(config-router-af)# aggregate-address <PREFIX> <MASK> [options]
```

- **IOS XR:**

```
RP/0/0/CPU0:R1(config-bgp-af)# aggregate-address <PREFIX/LENGTH> [options]
```

WARN

Discard Route: Upon successful creation of an aggregate route, the router automatically installs a **discard route (pointing to Null0)** for the aggregate prefix in its RIB. This is a loop prevention mechanism: if the specific routes contributing to the aggregate are withdrawn, this discard route ensures that traffic matching only the aggregate (and not any other more specific route) is dropped locally rather than potentially being looped.



Critical Prerequisite: For an aggregate to be generated, **at least one more specific prefix that falls within the aggregate range must exist in the BGP table** of the router performing the aggregation. These more specific prefixes could have been originated locally (via `network` or `redistribute` commands) or learned from other BGP peers.

Default Behavior of `aggregate-address`

If you configure `aggregate-address` with just the prefix and mask (no options):

1. **More Specific Routes Advertised:** By default, BGP will advertise **both** the newly created aggregate route **and** all the more specific "child" routes that contribute to it.

2. **Path Attributes of the Aggregate:**

    - **AS_PATH:** The aggregate route will typically have an AS_PATH containing only the AS number of the aggregating router.

    - **ORIGIN:** Usually `i` (IGP).

    - **NEXT_HOP:** `0.0.0.0` in the local BGP table (will be set to the router's IP when advertised).

    - **ATOMIC_AGGREGATE Attribute:** This Well-Known Discretionary attribute will likely be **added** to the aggregate. It signals to other BGP routers that the aggregate may not have complete path information because it summarizes routes that might have had diverse AS_PATHs.

- **AGGREGATOR Attribute:** An Optional Transitive attribute will be added, indicating the AS number and BGP Router ID of the router that performed the aggregation.

- **NO AS_SET created by default:** An AS_SET (an unordered list of ASNs from the component routes) is **not** created in the AS_PATH of the aggregate by default if the `as-set` option isn't used.

Common `aggregate-address` Options

Several options modify the behavior of the `aggregate-address` command:

1. `summary-only`

- **Purpose:** To suppress the advertisement of the more specific "child" routes that contribute to the aggregate. Only the aggregate route itself will be advertised.

- **Effect:** Routes suppressed by this command will appear in the local BGP table with an `s` status code, indicating they are suppressed.

- **Configuration:**
  - **IOS XE:** `aggregate-address <PREFIX> <MASK> summary-only`
  - **IOS XR:** `aggregate-address <PREFIX/LENGTH> summary-only`

2. `as-set`

- **Purpose:** To include AS_PATH information from the aggregated specific routes in the AS_PATH of the aggregate route. This is crucial for loop prevention when specific path information is lost due to summarization.

- **Effect:** It creates an **AS_SET** segment in the AS_PATH attribute of the aggregate. An AS_SET is an *unordered* list of all AS numbers present in the AS_PATHs of the specific routes being summarized. The aggregating router's AS number will still be prepended to this (as an AS_SEQUENCE).

  - Example AS_PATH with AS_SET: `65000 {65001,65002,65003}` (where 65000 is the aggregating AS).

- **Configuration:**
  - **IOS XE:** `aggregate-address <PREFIX> <MASK> as-set`
  - **IOS XR:** `aggregate-address <PREFIX/LENGTH> as-set`

3. `attribute-map <route-map-name>` (or `route-policy` in XR for similar effect)

- **Purpose:** By default, the aggregate route generates its own set of path attributes (e.g., AS_PATH typically contains only the local AS, ORIGIN is IGP, NEXT_HOP is self). The `attribute-map` option allows you to **set or modify specific path attributes** for the aggregate route.

- **How it Works:** You create a route map (IOS XE) or route policy (IOS XR) that contains `set` commands for various BGP attributes (e.g., `set origin`, `set local-preference`, `set community`, `set med`). This map/policy is then referenced by the `aggregate-address` command.

4. `suppress-map <route-map-name>`

- **Purpose:** To selectively suppress some of the more specific routes while allowing others to be advertised along with the aggregate. This offers more granular control than `summary-only`.

- **How it Works:** The `suppress-map` references a route map.
  - Specific routes **permitted** by this route map are **suppressed**.
  - Specific routes **denied** by this route map (or not matched by any permit clause) are **advertised** along with the aggregate.

5. `advertise-map <route-map-name>`

- **Purpose:** Used for **conditional advertisement** of the aggregate route.

- **How it Works:** The `advertise-map` references a route map which typically matches a set of specific prefixes (component routes). The aggregate route will only be generated and advertised if **at least one** of the specific routes permitted by the `advertise-map` exists in the BGP table.

BGP AS_PATH ACLs

BGP AS_PATH Access Lists & Regular Expressions

AS_PATH access lists use **regular expressions** to match patterns within the BGP AS_PATH attribute. They are commonly used with:

- `filter-list`: To permit or deny routes based on their AS_PATH content.

- **Route Maps/Policies**: To match routes based on their AS_PATH for the purpose of applying other policies (e.g., setting Weight, Local Preference, MED, Communities).

This means you can control routes based on:

1. The **originating AS** of the prefix.
2. The **first-hop AS** from which you received the prefix.
3. Any **transit ASes** the prefix has passed through.
4. Various combinations and sequences of ASes.

Configuration

- **IOS XE:**

```
R1(config)# ip as-path access-list <access-list-number> <permit | deny> <regular-expression>
```

- **IOS XR:** AS_PATH matching is done within Route Policy Language (RPL) using `as-path in <as-path-set-name>` or directly embedding regex patterns. An `as-path-set` is defined as:

```
RP/0/RP0/CPU0:R1(config)# as-path-set MY_AS_PATH_SET
RP/0/RP0/CPU0:R1(config-asp-set)# <regular-expression>
RP/0/RP0/CPU0:R1(config-asp-set)# end-set
!
RP/0/RP0/CPU0:R1(config)# route-policy MY_RPL
RP/0/RP0/CPU0:R1(config-rpl)# if as-path in MY_AS_PATH_SET then
! ... actions ...
RP/0/RP0/CPU0:R1(config-rpl-if)# endif
RP/0/RP0/CPU0:R1(config-rpl)# end-policy
```

Understanding AS_PATH Regular Expressions

The AS_PATH is treated as a **string of space-separated characters**. Regular expressions provide a powerful way to match patterns in this string.
Basic Regex Rules for AS_PATHs:

1. **AS Numbers as characters:** An AS number (e.g., `65001`) in your regex will match that literal AS number in the AS_PATH string.

2. `^` **(Caret): Beginning of the AS_PATH**
   - Matches the beginning of the AS_PATH string. `^65001` matches an AS_PATH that *starts* with AS 65001 (meaning 65001 is the first AS in the path, i.e., your direct eBGP peer's AS).

3. `$` **(Dollar Sign): End of the AS_PATH**
   - Matches the end of the AS_PATH string. `65001$` matches an AS_PATH that *ends* with AS 65001 (meaning 65001 is the originating AS).

4. `_` **(Underscore): AS Boundary / Delimiter**
   - Matches an AS delimiter, which can be:
     - A space between AS numbers.
     - A comma (used within AS_SETs, though matching specific AS_SET content is more complex).
   - Effectively, `_65001_` means "AS 65001 as a whole word/number."

5. `.` **(Dot): Any Single Character (Use with Caution)**
   - In general regex, `.` matches any single character. In the context of AS_PATH regex on Cisco routers, it's often used as a wildcard to match *any single AS number* when placed between delimiters (e.g., `_._` matches any single AS in a transit position). However, be aware that `.` technically matches just one character, not necessarily a multi-digit AS number.

6. `()` **(Parentheses): Grouping**
   - Groups parts of a regular expression. For example, `(_65001_)+` would match one or more occurrences of AS 65001. `(65001|65002)` matches either AS 65001 or AS 65002.

7. `[]` **(Square Brackets): Character Set (Less Common for Whole ASNs)**
    - Matches any single character *within* the brackets. `[123]` matches the character '1', '2', or '3'. It does **not** match "AS 1" or "AS 2" or "AS 3". For matching a set of AS numbers, use grouping and alternation: `(65001|65002|65003)`.

8. `*` **(Asterisk): Zero or More Occurrences**
    - Matches the preceding character or group zero or more times. For example, `65001(` `65002)*_65003_` means AS 65001, followed by zero or more occurrences of AS 65002, followed by AS 65003.

9. `+` **(Plus Sign): One or More Occurrences**
    - Matches the preceding character or group one or more times. For example, `(_65001_)+` matches one or more occurrences of AS 65001 in sequence (often used for prepended paths).

10. `|` **(Pipe/Alternation): OR Operator**
    - Matches either the expression before or the expression after it. Example: `^65001_|_65002$` matches paths starting with AS 65001 OR paths ending with AS 65002.

Examples

Given an AS_PATH string like: `^10_43_76_23_11_4$` (where `_` represents spaces, `^` the start, `$` the end)

- **Match Exactly:**
    - `^10_43_76_23_11_4$`

- **Match Routes Originated by AS 4:**
    - `_4$` (Matches AS 4 at the end of the path)
    - **Important Clarification:** Just `4$` would match any AS_PATH ending in the *character* '4' (like `34$` *or* `444$`). To match AS number 4 specifically as the last AS, `_4$` is correct.

- **Match Routes Received Directly from Peer AS 10:**
    - `^10_` (Matches AS 10 at the beginning of the path)

- **Match Routes Transiting Through AS 76:**
  - `76` (Matches AS 76 anywhere in the middle of the path)

- **Match Locally Originated Routes (Empty AS_PATH):**
  - `^$` (Matches an empty AS_PATH, before it's advertised to an eBGP peer and your own AS is prepended)

- **Match Routes Originated by AS 65001 or AS 65002:**
  - `_(65001|65002)$`

- **Match Routes Received from AS 65001 that has prepended itself (at least once, so total at least two 65001s at start):**
  - `^65001_65001_` or more generally `^65001_(_65001_)+`

- **Match Any Route (Permit All):**
  - `.*` (Dot matches any character, star means zero or more occurrences. This is a common "permit any" regex in many contexts).
  - Or simply an empty regex if the platform allows (often implies permit any).

More Advanced Matching Example:
Let's say you want to match a path that has AS 750 appearing one or more times, followed by AS 836:

- `(_750_)+_836_`
  - This would match `750` `836_`
  - This would match `750` `750_836_`
  - This would NOT match `836` or `75` `836_`

Regular expressions are incredibly powerful but require attention and testing. Start with simple patterns and build complexity as needed. Always test your AS_PATH ACLs thoroughly in a lab before deploying in a production network!

BGP Routes Filtering

Fundamentals of BGP Route Filtering

- **Location & Direction:** Filtering can be applied on any BGP router. It can be done for **inbound** updates (routes received from a neighbor) or **outbound** updates (routes advertised to a neighbor).
    - **Inbound Filtering:** Affects which routes from a neighbor are processed and considered for the local BGP table and subsequently for best path selection and RIB installation.
    - **Outbound Filtering:** Affects which routes from the local BGP table are advertised in UPDATE messages to a specific neighbor.
- **Policy Activation:** After implementing or changing a filtering policy, BGP needs to re-evaluate routes against this new policy. This requires a "reset" of the BGP session's advertisements:
    - **Hard Reset** `clear ip bgp <neighbor_ip>`
    - **Soft Reset / Route Refresh**

      `clear ip bgp <neighbor_ip> soft in/out`

      `clear ip bgp <neighbor_ip> in/out`

Filtering Criteria in BGP vs. IGPs

- **IGPs (OSPF, EIGRP, IS-IS):** Filtering is primarily based on **prefix and prefix length**. Tools like ACLs, prefix lists, and route maps are used for matching.
- **BGP:** Offers more granular control. You can filter based on:
    1. **Prefix and Prefix Length.**
    2. **BGP Path Attributes**

Tools for BGP Filtering

A. Filtering Based on Prefix / Prefix Length:

1. **Prefix Lists:**
    - **Purpose:** The **recommended and most precise tool** for matching IP prefixes and their lengths.
    - **Application:** Applied to BGP neighbors inbound or outbound.

- **IOS XE Configuration:**

```
R1(config)# ip prefix-list MY_PREFIX_LIST seq 10 deny 10.10.0.0/16
R1(config)# ip prefix-list MY_PREFIX_LIST seq 20 permit 0.0.0.0/0 le 32 // Permits all other routes

! Classical BGP CLI
R1(config-router)# neighbor <NEIGHBOR_IP> prefix-list MY_PREFIX_LIST <in | out>
! Upgraded BGP CLI (Address Family)
R1(config-router-af)# neighbor <NEIGHBOR_IP> prefix-list MY_PREFIX_LIST <in | out>
```

- **IOS XR Equivalent:** Prefix-sets are defined globally and then referenced within Route Policy Language (RPL).

```
RP/0/RP0/CPU0:R1(config)# prefix-set MY_PS deny 10.10.0.0/16
RP/0/RP0/CPU0:R1(config-pfx)# end-set // Assuming implicit permit any, or add permit 0.0.0.0/0 le 32
! Then use in RPL: if destination in MY_PS then drop ... endif
```

2. **Distribute Lists (using Standard or Extended ACLs):**

- **Purpose:** Can filter routes based on prefixes.
- **Standard ACL:** Matches only on the prefix address (network portion).
- **Extended ACL:** Can theoretically match prefix and length.

```
! IOS XE
R1(config)# access-list 1 deny 10.10.10.0 0.0.0.255 // Deny 10.10.10.0/24
R1(config)# access-list 1 permit any
R1(config-router)# distribute-list 1 <in | out> [interface] // Global to all neighbors
! OR
R1(config-router)# neighbor <NEIGHBOR_IP> distribute-list 1 <in | out> // Per neighbor
```

B. Filtering Based on Path Attributes:

1. **Filter Lists (for AS_PATH attribute filtering):**

- **Purpose:** Specifically designed to filter routes based on patterns in the **AS_PATH** attribute using AS_PATH access lists (which use regular expressions).

- **Configuration (IOS XE):**

    1. Create an AS_PATH access list:

```
R1(config)# ip as-path access-list 10 deny _65001_ // Deny routes transiting AS 65001
R1(config)# ip as-path access-list 10 permit .* // Permit all others (.* is a common regex for "any")
```

(Remember the implicit deny at the end of AS_PATH ACLs, so a final permit is usually needed.)

2. Apply to a neighbor:

```
! Classical BGP CLI
R1(config-router)# neighbor <NEIGHBOR_IP> filter-list 10 <in | out>
! Upgraded BGP CLI (Address Family)
R1(config-router-af)# neighbor <NEIGHBOR_IP> filter-list 10 <in | out>
```

- **IOS XR:** AS-path sets (defining regex) are used within RPL.

```
RP/0/RP0/CPU0:R1(config)# as-path-set DENY_AS65001_TRANSIT
RP/0/RP0/CPU0:R1(config-asp-set)# ios-regex '_65001_'
RP/0/RP0/CPU0:R1(config-asp-set)# end-set
!
RP/0/RP0/CPU0:R1(config)# route-policy FILTER_AS_PATH_RPL
RP/0/RP0/CPU0:R1(config-rpl)# if as-path in DENY_AS65001_TRANSIT then
RP/0/RP0/CPU0:R1(config-rpl-if)# drop
RP/0/RP0/CPU0:R1(config-rpl-if)# else
RP/0/RP0/CPU0:R1(config-rpl-else)# pass
RP/0/RP0/CPU0:R1(config-rpl-else)# endif
RP/0/RP0/CPU0:R1(config-rpl)# end-policy
! Apply route-policy to neighbor in/out
```

2. **Route Maps (IOS XE) / Route Policy Language (RPL on IOS XR):**

- **Purpose:** The most powerful and flexible tool for BGP policy implementation. They can match on various criteria (prefixes, path attributes) and then take actions (permit, deny, modify attributes).

- **Matching Capabilities for Filtering (IOS XE Route Map):**
  - `match ip address prefix-list <name>`

    `match ip address <ACL_NUMBER_OR_NAME>`
  - `match as-path <AS_PATH_ACL_NUMBER>`
  - `match community <COMMUNITY_LIST_NUMBER_OR_NAME> [exact-match]`
  - `match extcommunity <EXTCOMMUNITY_ACL_NUMBER_OR_NAME> [exact-match]`
  - `match mpls-label`

- **Unsupported `match` commands for direct attribute value checking (as conditions for filtering):** You generally don't directly `match local-preference <value>` or `match metric <MED_value>` as a primary condition in a route map in the same way you match a prefix or AS_PATH for filtering. Instead, you would match on prefixes/AS_PATH/communities and then *set* attributes like local-preference or MED, or filter based on these other matchable criteria.

- **IOS XR RPL:** Provides similar or even more extensive matching capabilities (e.g., `if destination in ...`, `if as-path in ...`, `if community matches-any ...`).

Filtering in IOS XR - Summary

- **Route Policy Language (RPL) is the primary tool.**
- **Prefix-sets** are used to define groups of prefixes (equivalent prefix lists).
- **AS-path-sets** are used to define AS_PATH regular expressions (equivalent to IOS XE `ip as-path access-list`).
- **Community-sets** are used to define communities for matching (equivalent to IOS XE `ip community-list`).
- RPL combines these sets with `if/then/else/elseif` logic to create powerful policies that can filter routes or modify attributes.

Preventing Your AS from Becoming a Transit AS (Advertising Local Routes Only)

A common BGP policy requirement is to advertise only the prefixes that originate within your own Autonomous System (AS) to your eBGP peers, and *not* advertise routes learned from one eBGP peer to another eBGP peer. This prevents your AS from becoming a transit path for traffic between other ASes, which you typically only want if you are an ISP intentionally providing transit services.

Here are two common methods to achieve this:

Option 1: Using a Distribute-List (with an ACL) Outbound

This method explicitly defines which prefixes your AS is allowed to advertise.

1. **Create an Access Control List (ACL)** that permits only your AS's own prefixes that you wish to advertise.

```
! Define an ACL to match your own prefixes
! Example: if your AS owns 150.10.0.0/16 and 160.20.20.0/24
access-list 10 permit 150.10.0.0 0.0.255.255
access-list 10 permit 160.20.20.0 0.0.0.255
! The ACL has an implicit "deny any" at the end
```

2. **Apply the Distribute-List Outbound to Your eBGP Neighbor(s):**

```
router bgp <YOUR_ASN>
 neighbor <PEER_IP_ADDRESS> remote-as <PEER_ASN>
 neighbor <PEER_IP_ADDRESS> distribute-list 10 out
```

- **How it Works:** The `distribute-list 10 out` command ensures that only prefixes explicitly permitted by `access-list 10` are advertised to the peer. Any routes learned from other BGP peers and present in your BGP table (which are not matched by a `permit` statement in ACL 10) will be filtered by the implicit deny at the end of the ACL and not advertised to this peer.

Option 2: Using a Filter-List (with an AS_PATH ACL) Outbound

This method filters advertisements based on the AS_PATH attribute, specifically allowing only locally originated routes (which have an empty AS_PATH before being advertised). This is a very common and effective technique.

1. **Create an IP AS_PATH Access List** that permits only routes with an empty AS_PATH (i.e., routes originated by the local AS).

```
! This AS_PATH ACL permits an empty AS_PATH (locally originated routes)
ip as-path access-list 1 permit ^$
! There's an implicit "deny all" for any non-empty AS_PATHs
```

   - `^` matches the beginning of the AS_PATH string.
   - `$` matches the end of the AS_PATH string.
   - `^$` together matches an empty string, which is characteristic of routes locally originated by the `network` or `aggregate-address` commands before they are advertised to an eBGP peer (at which point your own AS is prepended).

2. **Apply the Filter-List Outbound to Your eBGP Neighbor(s):**

```
router bgp <YOUR_ASN>
 neighbor <PEER_IP_ADDRESS> remote-as <PEER_ASN>
 neighbor <PEER_IP_ADDRESS> filter-list 1 out
```

- **How it Works:** The `filter-list 1 out` command ensures that only routes matching `ip as-path access-list 1` (i.e., your locally originated routes) are advertised to the peer. Any routes learned from other BGP peers will have a non-empty AS_PATH and will be denied by the implicit deny at the end of the AS_PATH access list, thus preventing them from being advertised to this peer.

Choosing Between Options:

- **Option 2 (AS_PATH filter-list with `^$`)** is generally simpler and more robust for the specific goal of advertising only locally originated prefixes and preventing transit AS behavior. It doesn't require you to list all your prefixes in an ACL.

- **Option 1 (Distribute-list)** gives you more explicit control if you want to advertise only a *subset* of your locally originated prefixes or have other complex prefix-based criteria.

By implementing one of these outbound filtering strategies with your eBGP peers, you can effectively control your BGP advertisements and ensure your AS does not become an unintended transit path. Remember to clear the BGP session after applying new outbound policies for them to take effect.

MP-BGP Introduction

MP-BGP (Multi-Protocol BGP)

When BGP was first designed (as BGP-4, defined in RFC 1771, now obsoleted by RFC 4271), its primary focus was IPv4 unicast routing. The UPDATE message contained a specific field for Network Layer Reachability Information (NLRI) tailored for IPv4 prefixes.

However, the architects anticipated the need for BGP to carry other types of routing information. Instead of redesigning the BGP UPDATE message for each new protocol, BGP was extended using **Multiprotocol Extensions for BGP-4 (MP-BGP)**, primarily defined in **RFC 4760** (which obsoletes the earlier RFC 2858).

How MP-BGP Works

MP-BGP introduces two new Optional Non-Transitive Path Attributes:

1. **MP_REACH_NLRI (Multiprotocol Reachable NLRI - Type Code 14):**
    - Used to advertise routes for address families other than IPv4 unicast.
    - It carries the next-hop information and the NLRI (prefixes) for a specific Address Family Identifier (AFI) and Subsequent Address Family Identifier (SAFI).

2. **MP_UNREACH_NLRI (Multiprotocol Unreachable NLRI - Type Code 15):**
    - Used to withdraw routes for address families other than IPv4 unicast.
    - It carries the NLRI (prefixes to be withdrawn) for a specific AFI/SAFI.

This approach allows the original BGP UPDATE message structure to remain largely intact, with new address families being supported by carrying their NLRI and next-hop information within these new attributes.

What Can Be Carried with MP-BGP?

MP-BGP can now carry a wide variety of "payloads," including:

1. IPv4 Unicast routes (can use original NLRI or MP_REACH_NLRI)
2. IPv6 Unicast routes
3. IPv4 Multicast routes (for multicast source discovery)

4. IPv6 Multicast routes
5. VPNv4 routes (for MPLS Layer 3 VPNs)
6. VPNv6 routes (for MPLS Layer 3 VPNs using IPv6)
7. VPLS (Virtual Private LAN Service) information
8. EVPN (Ethernet VPN) MAC/IP advertisement routes ...and more

---

The Concept of Address Families (AFs)

To manage these different types of reachability information, BGP configuration uses the concept of **Address Families**.

- **Default Behavior (Classic IOS/IOS XE):** When you enable BGP and configure a neighbor without specifying an address family, it typically defaults to operating only for the **IPv4 Unicast** address family.

- **IOS XR:** IOS XR is inherently address-family centric. You **must** configure and activate neighbors under specific address families. There are no default assumptions for neighbor activation outside of an AF context.

- **Explicit AF Configuration:** To use BGP for IPv6, MPLS VPNs, Multicast, etc., you must explicitly configure BGP to operate within that specific address family context and activate your neighbors for that AF.

When you configure an address family (e.g., IPv6 unicast) under a BGP process and activate a neighbor for it:

1. **NLRI Handling:** You are telling the BGP process to prepare to send and receive NLRI specific to that address family (e.g., IPv6 prefixes and next-hops) using the MP-BGP attributes (MP_REACH_NLRI and MP_UNREACH_NLRI).

2. **Capability Negotiation:** During the BGP OPEN message exchange with a peer, your router will advertise its capability to support this newly configured address family. The peer must also support and agree to exchange information for that address family for the peering to be fully active for that AF.

Configuration: Classic vs. Address Family Model (IOS XE)

Classic Configuration (Implicit IPv4 Unicast):

```
! Router BGP <ASN>
!  Neighbor X.X.X.X remote-as <ASN>
!  Network X.X.X.X mask Y.Y.Y.Y  // Advertises an IPv4 unicast prefix
```

Modern Address Family Configuration (Explicit):
This is the more versatile and clear way, essential for MP-BGP.

```
! Router BGP <ASN>
!  Neighbor X.X.X.X remote-as <ASN> // Establishes TCP session
!
!  address-family ipv4 unicast
!   Neighbor X.X.X.X activate        // Activates this neighbor for IPv4 unicast
!   Network X.X.X.X Mask Y.Y.Y.Y
!  exit-address-family
!
!  address-family ipv6 unicast       // Example for IPv6
!   Neighbor X.X.X.X activate        // Activates this neighbor for IPv6 unicast
!   Network A:B:C:D::/64
!  exit-address-family
```

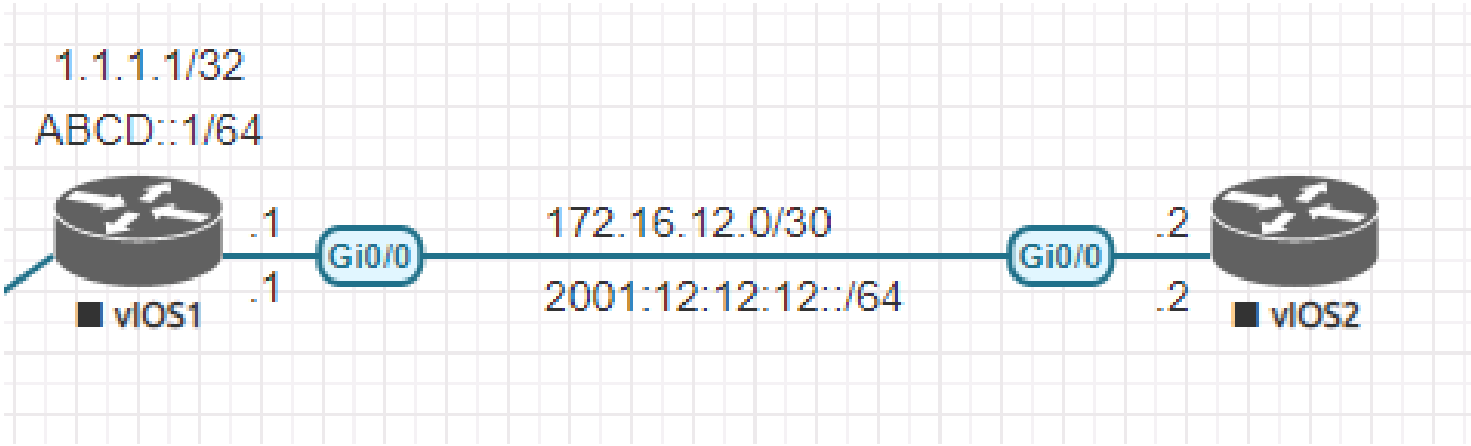
- **Clarity:** The address-family model clearly separates configurations for different types of NLRI.
- `neighbor <ip> activate` **under IPv4 AF (IOS XE):** if a neighbor is defined globally (outside any AF), IOS XE often activates it for IPv4 unicast by default. However, explicitly activating it under `address-family ipv4 unicast` is cleaner, more consistent with other AFs, and avoids ambiguity. It's not "extra" in the sense of being wrong; it's good practice.
- `bgp upgrade-cli` **(IOS XE):** Some IOS XE versions provide this command to automatically convert a classic BGP configuration to the newer address-family format.

BGP for IPv6

MP-BGP: IPv6 Address Family and Next-Hop Considerations

MP-BGP allows BGP to carry routing information for multiple network layer protocols. A key scenario is carrying IPv6 NLRI (Network Layer Reachability Information) over a BGP session that might be established using IPv4, or vice-versa, or over a dual-stack link.

The Next-Hop Problem in Mixed Environments



A fundamental rule in BGP is that the **next-hop address for a given address family must be an address of that same family.**

- If you are advertising an **IPv6 prefix** (e.g., `ABCD::/64`), the BGP NEXT_HOP attribute for that prefix **must be a valid IPv6 address**.

- If you are advertising an **IPv4 prefix** (e.g., `1.1.1.1/32`), the BGP NEXT_HOP attribute for that prefix **must be a valid IPv4 address**.

The Challenge:

If the BGP peering session itself is established over IPv4 (e.g., neighbor 172.16.12.2 remote-as 2), but you want to advertise IPv6 routes:

- When vIOS1 (in AS 1) sends an UPDATE for an IPv6 prefix like `ABCD::/64` to vIOS2 (in AS 2) over this IPv4-based BGP session, vIOS1 needs to specify a valid *IPv6* next-hop.

- By default, when advertising to an **eBGP peer**, BGP sets the next-hop to the IP address of the outgoing interface used for the peering session. If the peering is IPv4, the router would naturally try to set an IPv4 next-hop. This won't work for IPv6 NLRI.

- If the link between vIOS1 and vIOS2 *only* had IPv4 addresses, and vIOS1 tried to send an IPv6 prefix to vIOS2, vIOS2 would receive the advertisement, but without a valid IPv6 next-hop that it can resolve to forward IPv6 traffic back towards vIOS1's AS, the IPv6 route would be unusable. The traffic itself cannot be forwarded if the underlying link doesn't support IPv6 or if a valid IPv6 next-hop isn't provided.

HINT

Dual Stack Interfaces: configuring both IPv4 and IPv6 addresses on the physical link between BGP peers (a "dual-stack" link) is common and allows for proper next-hop advertisement for both address families.

Design Scenarios and Next-Hop Configuration

Scenario 1: BGP Peering over IPv4, Advertising Both IPv4 and IPv6 Routes

(vIOS1 peers with vIOS2 using vIOS2's IPv4 address 172.16.12.2)
The link G0/0 must have an IPv6 address configured on vIOS1 (e.g., `2001:12:12:12::1/64`) that vIOS2 can use as a next-hop for IPv6 traffic destined towards prefixes advertised by vIOS1.

```
! On vIOS1 (AS 1)
ipv6 unicast-routing
!
router bgp 1
 neighbor 172.16.12.2 remote-as 2 // IPv4 peering
 !
 address-family ipv4 unicast
  neighbor 172.16.12.2 activate
  network 1.1.1.1 mask 255.255.255.255 // IPv4 prefix, next-hop will be 172.16.12.1
 !
 address-family ipv6 unicast
  neighbor 172.16.12.2 activate
  network ABCD::/64
  neighbor 172.16.12.2 route-map SET_IPV6_NEXTHOP out // Set IPv6 next-hop
 !
!
route-map SET_IPV6_NEXTHOP permit 10
 set ipv6 next-hop global 2001:12:12:12::1 // vIOS1's G0/0 IPv6 address
!
```

- **Explanation:**
  - The BGP session is established over IPv4.
  - For IPv4 routes advertised under `address-family ipv4 unicast`, BGP will set the next-hop to vIOS1's IPv4 address on G0/0 (`172.16.12.1`) by default when sending to eBGP peer vIOS2.

- For IPv6 routes advertised under `address-family ipv6 unicast`, BGP by default **will not automatically pick the correct IPv6 interface address as the next-hop** when the peering session itself is IPv4. It needs to be explicitly told what IPv6 address to use.
- The `set ipv6 next-hop global <ipv6_address>` command in the outbound route-map ensures that vIOS1 advertises its G0/0 IPv6 address (`2001:12:12:12::1`) as the next-hop for the IPv6 prefix `ABCD::/64`. vIOS2 can then use this IPv6 next-hop.

Scenario 2: BGP Peering over IPv6, Advertising Both IPv4 and IPv6 Routes

(vIOS1 peers with vIOS2 using vIOS2's IPv6 address 2001:12:12:12::2)

```
! On vIOS1 (AS 1)
ipv6 unicast-routing
!
router bgp 1
 neighbor 2001:12:12:12::2 remote-as 2 // IPv6 peering
 !
 address-family ipv4 unicast
  neighbor 2001:12:12:12::2 activate
  network 1.1.1.1 mask 255.255.255.255
  neighbor 2001:12:12:12::2 route-map SET_IPV4_NEXTHOP out // Set IPv4 next-hop
 !
 address-family ipv6 unicast
  neighbor 2001:12:12:12::2 activate
  network ABCD::/64 // IPv6 prefix, next-hop will be 2001:12:12:12::1
 !
!
route-map SET_IPV4_NEXTHOP permit 10
 set ip next-hop 172.16.12.1 // vIOS1's G0/0 IPv4 address
!
```

- **Explanation:**
  - The BGP session is established over IPv6.
  - For IPv6 routes, BGP will correctly set the next-hop to vIOS1's IPv6 address on G0/0 (`2001:12:12:12::1`).
  - For IPv4 routes, an outbound route-map with `set ip next-hop <ipv4_address>` is needed to specify vIOS1's G0/0 IPv4 address (`172.16.12.1`) as the next-hop.

Scenario 3: Separate BGP Peering Sessions over IPv4 and IPv6 (Dual Peering)

This involves establishing two distinct BGP sessions between the routers, one using their IPv4 addresses and another using their IPv6 addresses .

```
! On vIOS1 (AS 1)
ipv6 unicast-routing
!
router bgp 1
 neighbor 172.16.12.2 remote-as 2 // IPv4 eBGP peer
 neighbor 2001:12:12:12::2 remote-as 2 // IPv6 eBGP peer (using global IPv6 address)
 !
 address-family ipv4 unicast
  neighbor 172.16.12.2 activate
  network 1.1.1.1 mask 255.255.255.255
 !
 address-family ipv6 unicast
  neighbor 2001:12:12:12::2 activate // Activate the IPv6 peer under IPv6 AF
  network ABCD::/64
 !
```

- **Explanation:**
  - Two separate BGP sessions are maintained.
  - The IPv4 session (to `172.16.12.2`) handles IPv4 NLRI. The next-hop for IPv4 routes will be `172.16.12.1`.
  - The IPv6 session (to `2001:12:12:12::2`) handles IPv6 NLRI. The next-hop for IPv6 routes will be `2001:12:12:12::1`.
  - No route-maps are needed here to set next-hops for the primary address family of each session.

Supporting Only IPv6 in BGP

If you want the router to *only* participate in BGP for the IPv6 address family and not IPv4:

- By default, even if you only configure the IPv6 address family, BGP on IOS XE might still attempt to use IPv4 capabilities or assume IPv4 peering if not explicitly disabled.
- To ensure BGP only signals IPv6 capability and doesn't attempt IPv4:

```
! On vIOS1 (AS 1) - Example for IOS XE
ipv6 unicast-routing
!
router bgp 1
 no bgp default ipv4-unicast // Disables default activation of IPv4 Unicast AF
 neighbor 2001:12:12:12::2 remote-as 2 // Define the IPv6 peer
 !
 address-family ipv6 unicast
  neighbor 2001:12:12:12::2 activate // Activate the peer for IPv6
  network ABCD::/64
 !
```

- **`no bgp default ipv4-unicast`**: This command prevents the BGP router from automatically activating the IPv4 unicast address family or assuming IPv4 capabilities for peers defined at the global BGP level.

- With this, the router will only form BGP sessions and exchange NLRI for the explicitly configured and activated address families (IPv6 unicast in this case). It will advertise only IPv6 capability in its OPEN message. If you later need IPv4, you would have to explicitly create the `address-family ipv4 unicast` section and activate neighbors under it.

BGP Config Optimization

Simplifying Configuration

As BGP deployments grow, configuring and managing individual neighbors with many shared policies can become cumbersome and error-prone. Peer groups (in IOS XE) and configuration templates/groups (in IOS XR) address this by allowing you to define a common set of configurations once and apply them to multiple neighbors.

Why Do We Need Peer Groups / Templates?

Simplified Configuration & Reduced Time

- Instead of repeating numerous configuration lines for each neighbor that shares common policies (like `remote-as` for iBGP, `update-source`, route maps, filter lists, timers, etc.), you define these policies once in a group/template.
- Neighbors are then made members of this group, inheriting all its common configurations. This drastically reduces the number of configuration lines and the time spent on setup and modification.

Better CPU Utilization

- When BGP generates UPDATE messages, without peer groups, it process policies and generate the update independently for each peer, even if many peers share the same outbound policies.
- With peer groups groups, BGP can generate the UPDATE message (including policy processing) **once** for the group and then replicate this pre-processed update to all members of that group that share the exact same outbound policy. This significantly reduces CPU load, especially during full table advertisements or when many routes change.

BGP Peer Groups in IOS XE

Concept: You create a named peer group, configure common session parameters and policies on this group "template," and then assign individual neighbors to this peer group.

Configuration Example:

Imagine you have 100 iBGP neighbors in AS 100, all needing update-source Loopback0, an inbound route-map R1_IN, an inbound filter-list 1, and an outbound distribute-list 1_OUT.

- **Without Peer Group (for one neighbor):**

```
router bgp 100
 neighbor 1.1.1.1 remote-as 100
 neighbor 1.1.1.1 update-source Loopback0
 neighbor 1.1.1.1 route-map R1_IN in
 neighbor 1.1.1.1 filter-list 1 in
 neighbor 1.1.1.1 distribute-list 1_OUT out
! ...and repeat for 99 more neighbors...
```

- **With Peer Group:**

1. Define the peer group and apply common policies:

```
router bgp 100
 neighbor IBGP_PEERS peer-group
 neighbor IBGP_PEERS remote-as 100
 neighbor IBGP_PEERS update-source Loopback0
 neighbor IBGP_PEERS route-map R1_IN in
 neighbor IBGP_PEERS filter-list 1 in
 neighbor IBGP_PEERS distribute-list 1_OUT out
```

2. Assign actual neighbors to the peer group:

```
router bgp 100
 neighbor 1.1.1.1 peer-group IBGP_PEERS
 neighbor 2.2.2.2 peer-group IBGP_PEERS
 ! ...and so on for all 100 neighbors...
```

WARN

- **Outbound Policies:** Policies applied outbound to the peer group template (e.g., `route-map XYZ out`) generally **cannot be overridden** on a per-neighbor basis for members of that group. All members inherit and use the group's outbound policy. This is linked to the CPU optimization for update generation.

- **Inbound Policies:** Policies applied inbound to the peer group template **can often be overridden** by configuring a specific inbound policy directly under a neighbor who is a member of the group. The member-specific policy takes precedence.

Configuration Groups/Templates in IOS XR

IOS XR evolves the peer group concept into a more structured and powerful system of configuration templates. Instead of a single "peer-group" type, XR offers several specialized group types that can be combined:

1. **Neighbor Group (`neighbor-group <name>`):**
    - This is the most analogous to an IOS XE peer group. It acts as a container that can inherit from session groups and AF groups, or have policies directly configured. It allows grouping of both session parameters and address-family policies.

2. **Session Group (`session-group <name>`):**
    - **Purpose:** To define and apply common BGP *session establishment parameters* (e.g., `remote-as`, `password`, `update-source`, timers, `ebgp-multihop`, `transport connection-mode passive`).
    - It does **not** contain address-family specific policies (like route policies).

3. **AF Group (`af-group <name> address-family <afi-safi>`):**
    - **Purpose:** To define and apply common *address-family specific policies* (e.g., `route-policy in/out`, `maximum-prefix`, `default-originate`, `next-hop-self`).
    - It does **not** contain session establishment parameters.

How they work in IOS XR:

- You define these groups (templates).

- For an individual neighbor, you can then `use session-group <name>` to inherit session parameters and, under the neighbor's address-family configuration, `use af-group <name>` to inherit address-family policies.

- A `neighbor-group` can be configured with its own parameters or can `use session-group` and `use af-group` itself, and then individual neighbors can `use neighbor-group`. This provides a hierarchical and modular way to build configurations.

IOS XR Configuration Example (using Session Group and AF Group):

```
router bgp 1000
 !
 address-family ipv4 unicast
  redistribute connected // Example global AF config
 !
 af-group MY_AF_POLICIES address-family ipv4 unicast
  route-policy CUSTOMER_IN in
  maximum-prefix 100 75
  route-policy DEFAULT_ROUTE_OUT out
  default-originate
 !
 session-group CUSTOMER_SESSION_PARAMS
  remote-as 65111
  password encrypted <some_password>
  session-open-mode passive-only // Equivalent to transport connection-mode passive
 !
 neighbor 1.2.3.5
  use session-group CUSTOMER_SESSION_PARAMS
  description *** Customer A via Session and AF Group ***
  address-family ipv4 unicast
   use af-group MY_AF_POLICIES
  !
 neighbor 1.2.3.6
  use session-group CUSTOMER_SESSION_PARAMS
  description *** Customer B via Session and AF Group ***
  address-family ipv4 unicast
   use af-group MY_AF_POLICIES
  !
```

This structure makes the configuration highly organized and scalable. If a session parameter needs to change for all customers using `CUSTOMER_SESSION_PARAMS`, you change it once in the session group.

The Goal of Peer Groups and Templates

Whether using IOS XE peer groups or IOS XR's configuration groups/templates, the primary objectives are:

1. **Scalability:** Make BGP configurations easier to read, manage, and modify as the number of peers grows.

2. **Resource Optimization:** Reduce CPU load during BGP update generation by processing policies once for a group of peers sharing identical outbound characteristics.

BGP Route Reflector

Solving iBGP Scaling Issue

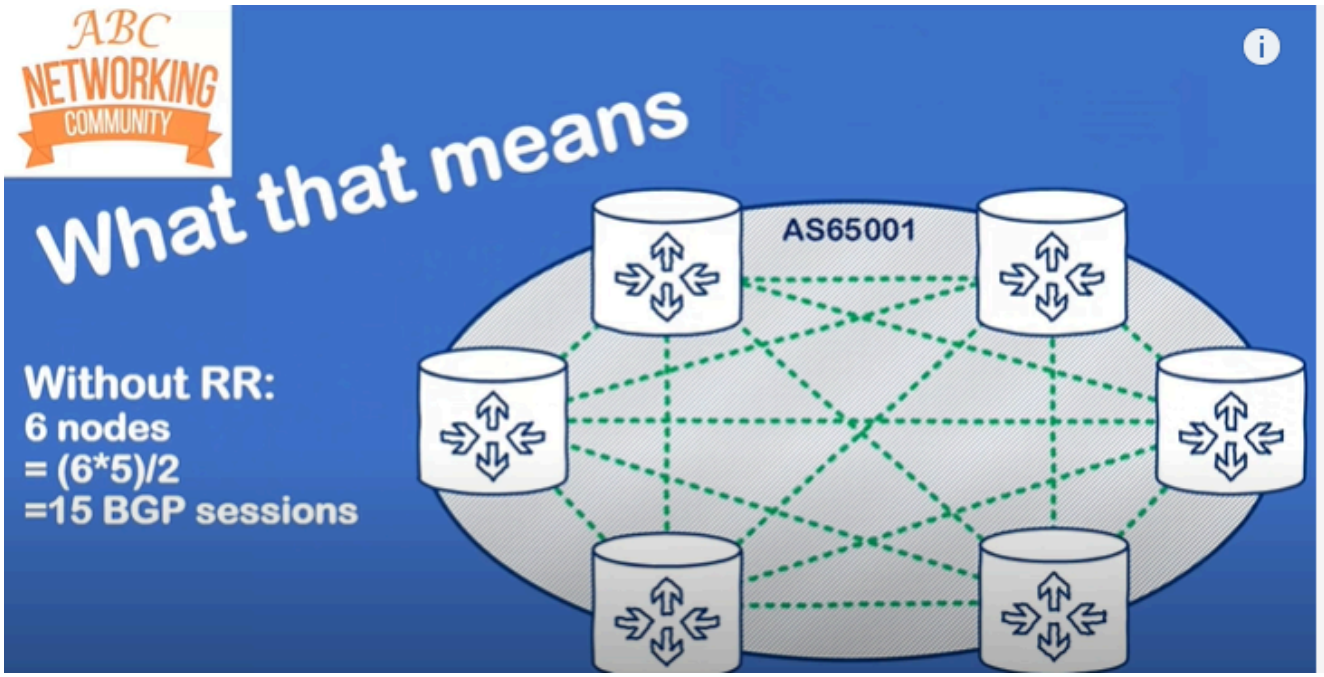
The Problem: iBGP Split Horizon and Scaling

1. **iBGP Split-Horizon Rule:** By default, a BGP speaker **will not advertise a route learned from one iBGP peer to another iBGP peer**.

2. **Reason for the Rule:** This rule is necessary for loop prevention within an AS because the **AS_PATH attribute is not modified when routes are passed between iBGP peers**. Without this rule, routes could loop indefinitely inside the AS.

3. **Consequence of the Rule:** To ensure all iBGP speakers within an AS receive all iBGP-learned routes (which typically originate from eBGP peers at the edge of the AS), one of the following traditional solutions was needed:

    - **Full Mesh:** Every iBGP speaker peers directly with every other iBGP speaker. This quickly becomes unmanageable in terms of configuration and resource consumption as the number of routers grows.



    - **BGP Confederations:** Divides a large AS into smaller sub-ASes with eBGP-like behavior between them. More complex to design and manage than Route Reflectors for many scenarios.

- **Route Reflectors:** The most common solution.

How Route Reflectors Solve the Scaling Problem

A Route Reflector is an iBGP router that is allowed to "reflect" (re-advertise) iBGP-learned routes to other iBGP peers, thereby relaxing the strict iBGP split-horizon rule under specific conditions.

- **RR Clients:** These are iBGP peers of the Route Reflector that the RR is configured to "serve." Clients typically only need to peer with their designated RR(s).

- **RR Non-Clients:** These are other iBGP peers of the Route Reflector that are not configured as clients. These are typically other RRs (for redundancy or hierarchy) or routers that still require a full mesh with the RR and other non-clients.

- **Cluster:** An RR and its clients form a "cluster." To prevent loops within a cluster or between clusters, RRs use BGP attributes like `CLUSTER_LIST` and `ORIGINATOR_ID`.

Reflection Rules (Simplified):

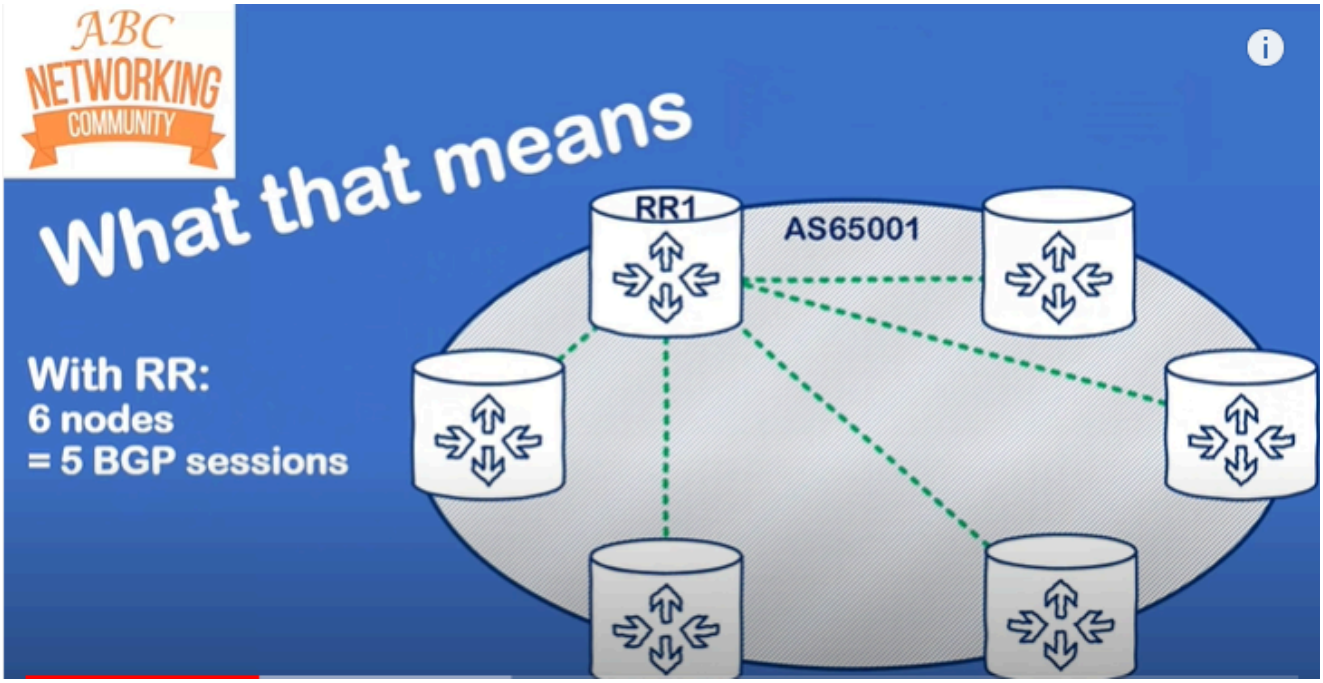
An RR modifies the iBGP split-horizon rule as follows:

1. **Route from an eBGP Peer:** When an RR learns a route from an eBGP peer, it reflects that route to **all its clients and all its non-client iBGP peers.**

2. **Route from a Non-Client iBGP Peer:** When an RR learns a route from a non-client iBGP peer, it reflects that route to **all its clients ONLY**. It does *not* reflect it to other non-client iBGP peers.

3. **Route from a Client iBGP Peer:** When an RR learns a route from one of its clients, it reflects that route to:
    - **All other clients** (except the originating client).
    - **All its non-client iBGP peers.**

Best Practice: A common design is to make all regular iBGP speakers clients of one or more RRs. The RRs themselves might peer with each other as non-clients or in a hierarchy.



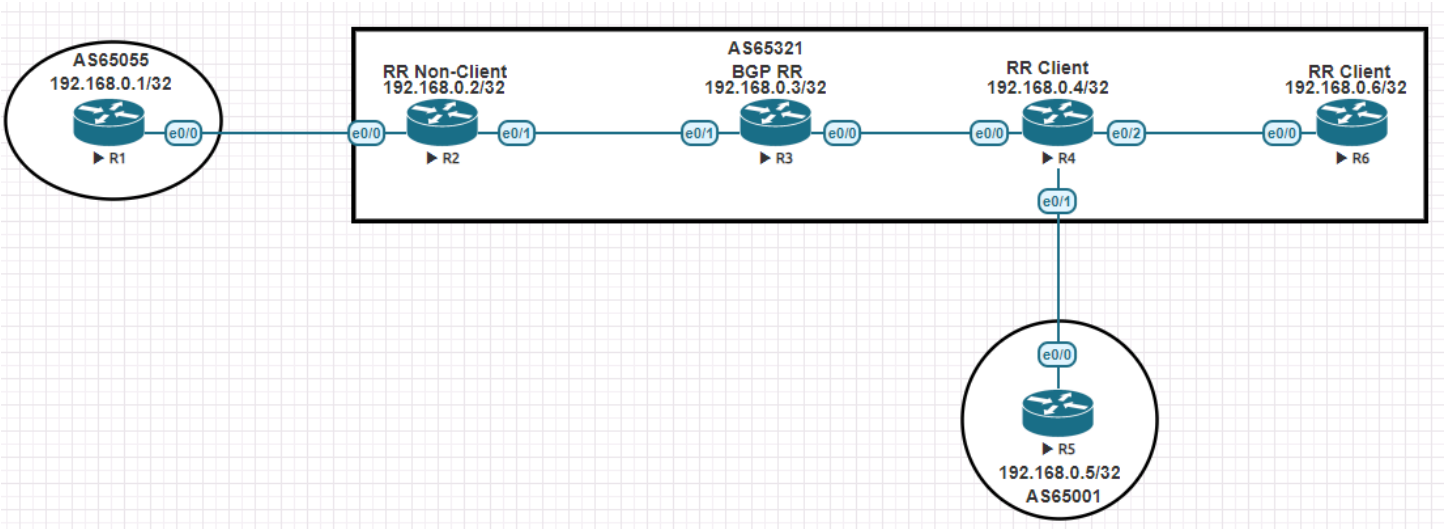
Next-Hop Behavior with Route Reflector

The term "reflector" is used because, by default, when an RR reflects a route, it **does not change the BGP NEXT_HOP attribute.**

- **Implication:** If an RR reflects a route originally learned from an eBGP peer (with the eBGP peer's IP as the next-hop), or from another iBGP speaker that didn't set `next-hop-self`, that original next-hop is preserved in the reflected route.

- **IGP Reachability:** All iBGP speakers (including clients receiving reflected routes) **must have IGP reachability to this BGP NEXT_HOP** for the route to be usable.

Route Reflector Lab Scenario and Configuration



Route Propagation Example:

1. R1 (AS55055) sends an eBGP update for `192.168.0.1/32` to R2.

2. R2 (AS65321, non-client of R3) learns this. R2 advertises this route to R3 (RR) via iBGP. R2 should use `next-hop-self` towards R3 so R3 sees the next-hop as R2's IP.

3. R3 (RR) receives the route from R2 (non-client). R3 reflects this route to its clients: R4 and R6. The next-hop will be R2's IP. R4 and R6 need IGP reachability to R2's IP.

4. R5 (AS65001) sends an eBGP update to R4.

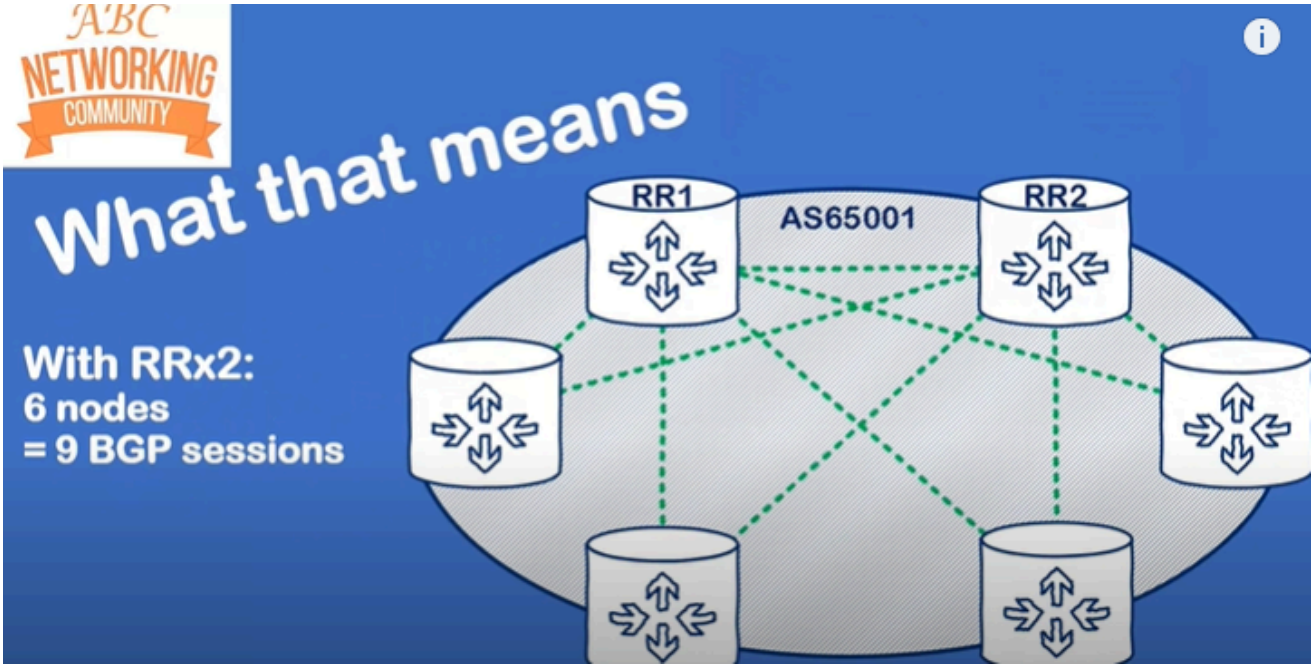
5. R4 (client of R3) learns this. R4 advertises this route to R3 (RR) via iBGP (using `next-hop-self` if needed).

6. R3 (RR) receives the route from R4 (client). R3 reflects this route to:
    - Its other client, R6.
    - Its non-client peer, R2.

Redundancy:

RRs are critical points. For redundancy, you typically deploy at least two RRs.

- Clients will peer with both RRs.
- The RRs themselves will usually have an iBGP peering between them (often as non-clients of each other, or configured in a specific RR hierarchy). This still results in far fewer iBGP sessions than a full mesh.



Configuring a Route Reflector and its Clients

The configuration is straightforward. Only the Route Reflector needs to know which of its iBGP peers are clients. The clients are configured as normal iBGP peers to the RR.

- **IOS XE (on the Route Reflector router, e.g., R3):**

```
R3(config)# router bgp <YOUR_AS>
R3(config-router)# address-family ipv4 unicast // Or other relevant AF
R3(config-router-af)# neighbor <CLIENT_IP_R4> route-reflector-client
R3(config-router-af)# neighbor <CLIENT_IP_R6> route-reflector-client
! R2 would be a normal iBGP peer (non-client) to R3
R3(config-router-af)# neighbor <NON_CLIENT_IP_R2> remote-as <YOUR_AS>
R3(config-router-af)# neighbor <NON_CLIENT_IP_R2> update-source <R3_LOOPBACK>
```

- **IOS XR (on the Route Reflector router, e.g., R3):**

```
RP/0/RP0/CPU0:R3(config)# router bgp <YOUR_AS>
RP/0/RP0/CPU0:R3(config-bgp)# neighbor <CLIENT_IP_R4>
RP/0/RP0/CPU0:R3(config-bgp-nbr)# remote-as <YOUR_AS>
RP/0/RP0/CPU0:R3(config-bgp-nbr)# address-family ipv4 unicast
RP/0/RP0/CPU0:R3(config-bgp-nbr-af)# route-reflector-client
! Repeat for other clients like R6
!
RP/0/RP0/CPU0:R3(config-bgp)# neighbor <NON_CLIENT_IP_R2>
RP/0/RP0/CPU0:R3(config-bgp-nbr)# remote-as <YOUR_AS>
RP/0/RP0/CPU0:R3(config-bgp-nbr)# address-family ipv4 unicast
! No 'route-reflector-client' for R2, making it a non-client
```

RRs: Control Plane vs. Data Plane Function

A very important point to remember about Route Reflectors is their primary role in the BGP architecture:

- **Control Plane Function:** Route Reflectors are fundamentally **control plane devices**. Their main job is to receive BGP UPDATE messages (routes) from iBGP peers and "reflect" or re-advertise these routes to other iBGP peers according to the RR rules (client-to-client, client-to-non-client, non-client-to-client). They are responsible for propagating reachability information throughout the AS without requiring a full iBGP mesh.

- **Data Plane Path:** The Route Reflector itself **does not necessarily have to be in the data path** for the traffic flowing towards the prefixes it reflects.
    - When an RR reflects a route, it typically does not change the BGP NEXT_HOP attribute.
    - Therefore, the actual forwarding path for data traffic is determined by the IGP routing within the AS towards the *original BGP next-hop* associated with the reflected route.
    - It's common for RRs to be dedicated devices that are powerful in terms of CPU and memory for BGP processing but might not have the high-throughput data plane interfaces of core forwarding routers.
    - While an RR *can* also be a transit router in the data path (e.g., if it's a core router performing RR duties), its function *as an RR* is purely about route advertisement in the control plane. The decision to place an RR in the data path is a separate network design choice.

This distinction ensures that the scalability solutions for the iBGP control plane (like RRs) do not impose constraints or bottlenecks on the actual data forwarding paths, which are determined by the IGP and the final BGP next-hop resolution.

BGP Confederations

Conquering iBGP Complexity

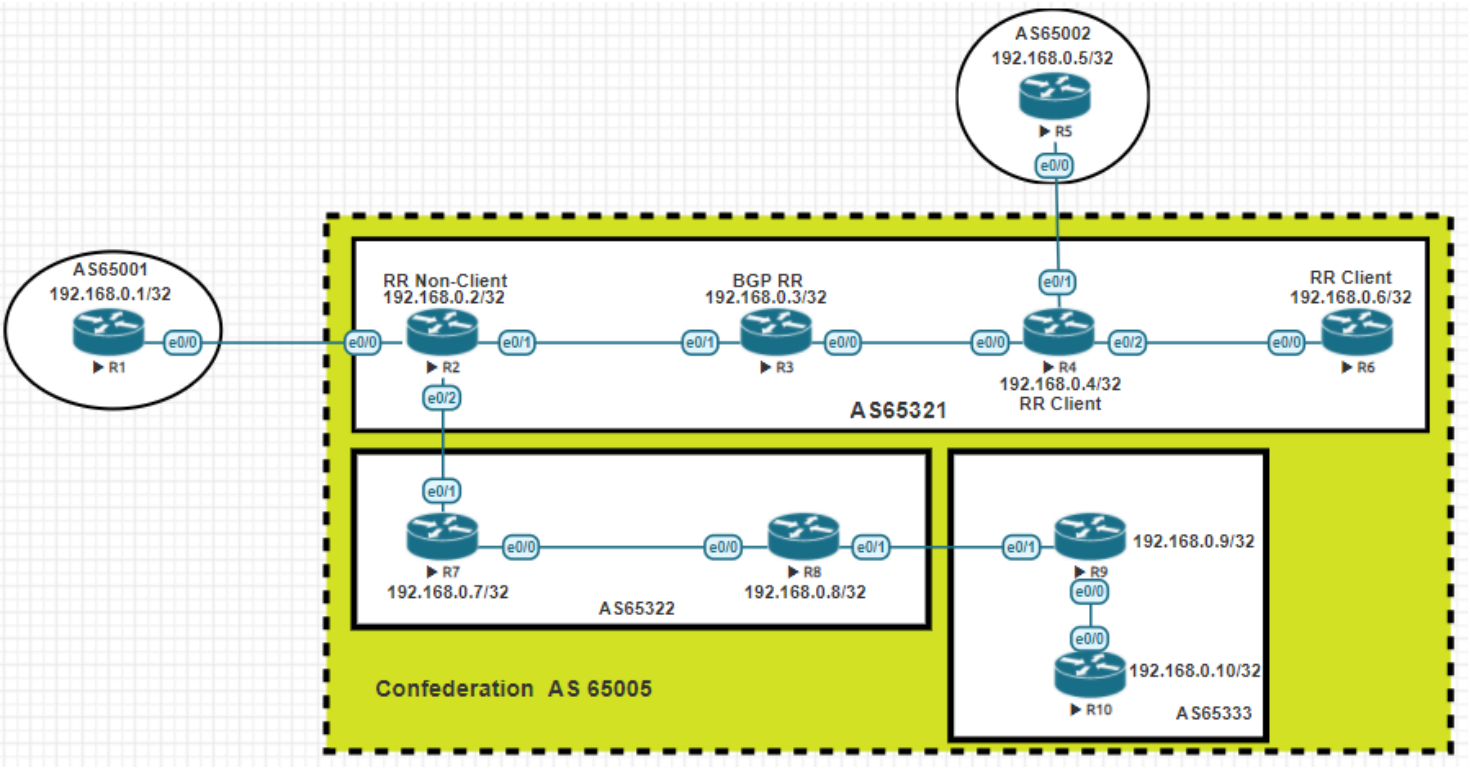
When an Autonomous System (AS) grows very large, a full iBGP mesh becomes impractical. While Route Reflectors (RRs) are one solution, BGP Confederations offer an alternative approach by dividing the large AS into smaller, more manageable sub-ASes.

Why Do We Need Confederations?

- **iBGP Scaling Problem:** As discussed, the iBGP split-horizon rule (a route learned from one iBGP peer is not advertised to another iBGP peer) necessitates a full mesh of iBGP peerings for all routers within an AS to receive all iBGP-learned routes. This scales poorly.

- **Alternative to Full Mesh/RRs:** Confederations provide a way to reduce the number of required iBGP peerings by breaking down the single large AS into multiple smaller **member-ASes**.

- **External Appearance:** From the perspective of routers **outside** the confederation, the entire group of member-ASes appears as a **single, larger AS**, identified by a "confederation identifier" (which is a public AS number). The internal sub-AS structure is hidden.



Terminology

- **Confederation:** The overall logical grouping of member-ASes that presents itself as a single AS to the outside world.

- **Confederation Identifier (Confed ID):** The AS number that represents the entire confederation to external BGP peers. This is typically the publicly registered AS number.

- **Member-AS (Sub-AS):** One of the smaller Autonomous Systems within the confederation. These member-ASes usually use private AS numbers (e.g., 64512-65534) or ASNs from the 32-bit private range.

Peering Rules Within a BGP Confederation

The behavior of BGP peerings changes depending on where the peers are located:

1. **Within a Member-AS:**
    - Peering between routers *inside the same member-AS* follows **standard iBGP rules**.
    - This means a full mesh is required within that member-AS, or Route Reflectors must be used *within that member-AS* to propagate routes.

2. **Between a Member-AS and an External AS (Outside the Confederation):**
    - Peering between a router in a member-AS and a router in a completely external AS follows **standard eBGP rules**.
    - The AS_PATH will show the Confederation ID, not the internal member-AS numbers, to the external peer.

3. **Between Different Member-ASes (Intra-Confederation Peering):**

    - This is where special "confederation eBGP" (sometimes called CEBGP or EBGP-Confed) rules apply. It's a hybrid behavior:

        - **AS_PATH:** The sending router **prepends its own member-AS number** to the AS_PATH when advertising to a peer in another member-AS. These member-AS numbers are typically enclosed in parentheses `()` in AS_PATH displays and are stripped out when the route is advertised outside the entire confederation.

        - **Next-Hop:** By default, the **NEXT_HOP attribute is NOT changed** when advertising routes between member-ASes (similar to iBGP). This means an IGP must provide reachability to the original next-hop across member-AS boundaries, or `next-hop-self` must be used.

        - **LOCAL_PREF & MED:** These attributes **ARE exchanged** between member-ASes (similar to iBGP). This allows for consistent policy application across the entire confederation.

- **TTL:** Typically, these sessions behave like eBGP regarding TTL (default 1 if directly connected, `ebgp-multihop` needed if not).

- **Loop Prevention:** The AS_PATH, now including member-AS numbers, prevents loops between member-ASes.

Do Confederations Replace Route Reflectors?

No, they are complementary solutions.

- Confederations address the iBGP scaling issue at a macro level by breaking the large AS into smaller domains (member-ASes).

- **Within each member-AS**, if it's still large enough to require many iBGP peerings, **Route Reflectors can (and often should) be used** to avoid a full iBGP mesh *inside* that member-AS.

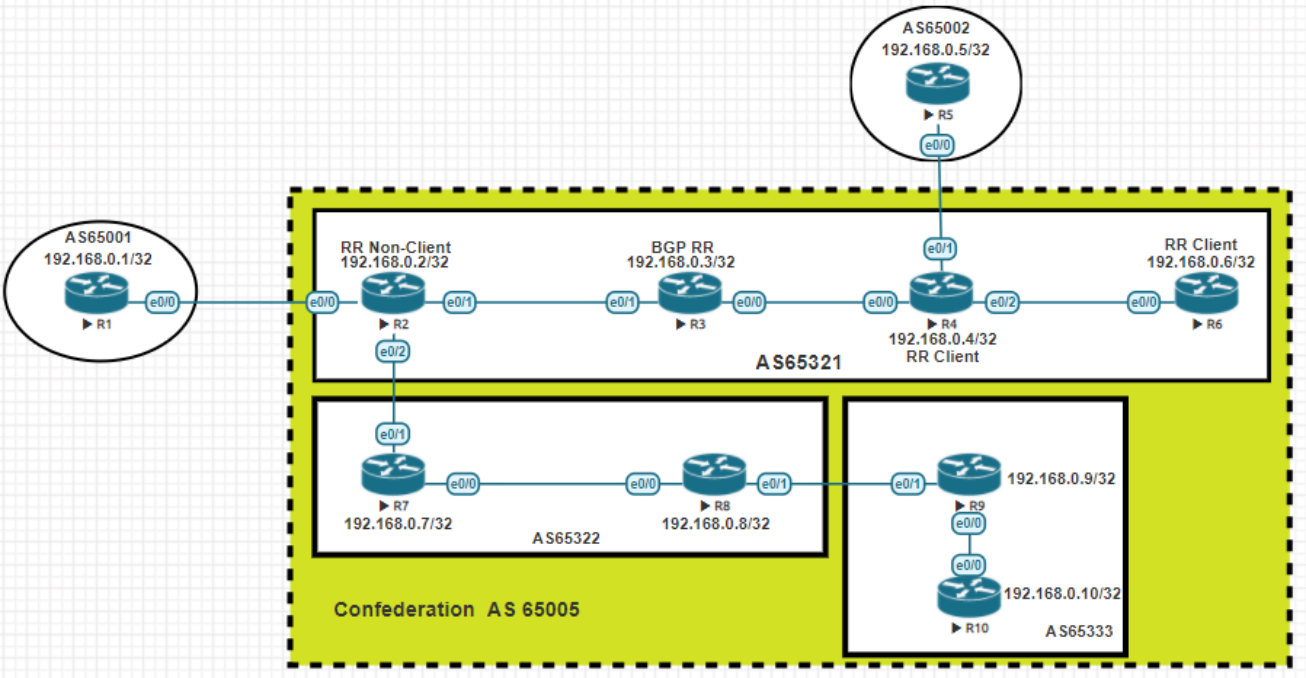
Configuring BGP Confederations

Configuration involves defining the member-AS the router belongs to, the overall confederation identifier, and which other member-ASes are part of the confederation (for routers peering between member-ASes).

1. `router bgp <MEMBER_AS_NUMBER>`:
    - Each router within the confederation is configured with its *actual* member-AS number (often a private ASN).

2. `bgp confederation identifier <PUBLIC_CONFEDERATION_ID_ASN>`:
    - **Mandatory on all routers** within the confederation.
    - This defines the single, public AS number that the entire confederation will present to the outside world.

3. `bgp confederation peers <MEMBER_AS_1> <MEMBER_AS_2> ...`:
    - Configured on routers that will form **intra-confederation peerings** (i.e., peer with routers in *other member-ASes* within the same confederation).
    - This command tells the router to treat peers in these listed ASNs using the special intra-confederation rules (e.g., passing LOCAL_PREF and MED, but treating it as an eBGP-like hop for AS_PATH).



R2 is in member-AS 65321.

- The overall confederation ID is 65005.
- R2 peers with other member-ASes 65322 and 65333 (so these are listed in `bgp confederation peers`).
- Its peering with R3 (192.168.0.3) is iBGP within member-AS 65321.
- Its peering with R7 (192.168.0.7 in member-AS 65322) is an intra-confederation peering.
- Its peering with R1 (192.168.12.1 in external AS 65001) is a standard eBGP peering.

```
R2#show running-config | section bgp
router bgp 65321
 bgp log-neighbor-changes
 bgp confederation identifier 65005
 bgp confederation peers 65322 65333
 neighbor 192.168.0.3 remote-as 65321
 neighbor 192.168.0.3 update-source Loopback0
 neighbor 192.168.0.7 remote-as 65322
 neighbor 192.168.0.7 ebgp-multihop 255
 neighbor 192.168.0.7 update-source Loopback0
 neighbor 192.168.12.1 remote-as 65001
 !
 address-family ipv4
  network 192.168.0.2 mask 255.255.255.255
  neighbor 192.168.0.3 activate
  neighbor 192.168.0.3 next-hop-self
  neighbor 192.168.0.7 activate
  neighbor 192.168.0.7 next-hop-self
  neighbor 192.168.12.1 activate
 exit-address-family
Exit
```

Confederations offer a structured way to scale iBGP by dividing administrative domains while maintaining a unified external AS appearance.

BGP Route Dampening

Punish Unstable Routes

Route flapping (routes that repeatedly become available and then unavailable) can cause significant instability in BGP tables and the wider internet, leading to increased CPU load on routers and potential routing inconsistencies. BGP Route Dampening is a mechanism designed to mitigate this by penalizing frequently flapping routes.

Why Use BGP Route Dampening?

- **Improve Stability:** The primary goal is to reduce the propagation of unstable route information, leading to a more stable BGP routing environment.

- **Reduce Processing Overhead:** Constantly processing UPDATE messages for flapping routes consumes router CPU and memory. Dampening reduces this.

How Does Route Dampening "Punish" Routes?

When a route is dampened, the router will:

1. **Stop Using the Route:** The dampened route is not considered for best path selection and is not installed in the RIB.

2. **Stop Advertising the Route:** The route is not advertised to BGP peers. This "punishment" lasts for a period, giving the unstable route time to stabilize.

Key Parameters of BGP Route Dampening

Route dampening behavior is controlled by several configurable parameters:

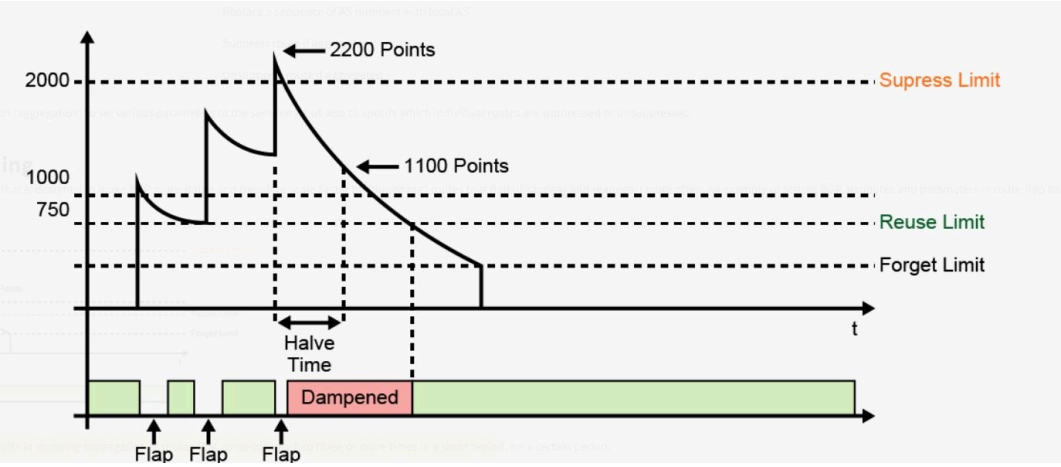
1. **Penalty:**
    - Each time a route flaps (goes down and comes back up, or attributes change causing a withdrawal and re-advertisement), a fixed penalty value is added to an accumulated penalty for that route.
    - **Default Penalty per Flap: 1000**

2. **Suppress Limit (or Suppress Threshold):**
    - This is a threshold for the accumulated penalty.
    - If a route's accumulated penalty **exceeds** this suppress limit, the route is **dampened** (suppressed).

3. **Half-Life Time:**
    - This timer determines how quickly a route's accumulated penalty decays if the route remains stable (i.e., does not flap).
    - For every half-life period that passes without a flap, the route's accumulated penalty is reduced by half.
    - **Example:** If the suppress limit is 3000 (meaning 3 flaps would suppress the route initially if the penalty per flap is 1000), and the half-life is 5 minutes:
        - After 3 flaps, penalty = 3000 (suppressed).
        - If stable for 5 mins, penalty decays to 1500.
        - If stable for another 5 mins, penalty decays to 750.

4. **Reuse Limit (or Reuse Threshold):**
    - This is a lower penalty threshold.
    - Once a dampened route's accumulated penalty decays **below** this reuse limit (due to stability over half-life periods), the route becomes **un-dampened** and is eligible again for best path selection and advertisement.
    - If it flaps again after being un-dampened, the penalty of 1000 is added, and it might quickly exceed the suppress limit again.

5. **Maximum Suppress Limit (Max Suppress Time):**
    - This defines the **absolute maximum time** a route can remain suppressed, regardless of how high its accumulated penalty is or how slowly it decays.
    - If a route has been suppressed for this maximum duration, it will be un-suppressed and considered for use again, even if its penalty hasn't decayed below the reuse limit. This prevents a persistently (but perhaps very infrequently) flapping route from being suppressed indefinitely.



Configuring BGP Route Dampening

Route dampening is typically configured globally within the BGP process or per address family.

- **IOS XE:**

```
R1(config)# router bgp <ASN>
! For Address Family CLI (recommended)
R1(config-router)# address-family ipv4 unicast
R1(config-router-af)# bgp dampening [<half-life> <reuse-threshold> <suppress-threshold> <max-suppress-time>] [route-map <map-name>]
```

    - If no parameters are specified, Cisco IOS uses default values (e.g., half-life 15 min, reuse 750, suppress 2000, max-suppress-time 60 min – defaults can vary by IOS version).
    - The `route-map <map-name>` option allows for **conditional dampening**, where dampening parameters are applied only to routes matching the route map.

```
R4(config-router-af)#bgp dampening ?
  <1-45>     Half-life time for the penalty
  route-map  Route-map to specify criteria for dampening
  <cr>

R4(config-router-af)#bgp dampening 1 ?
  <1-20000>  Value to start reusing a route
  <cr>

R4(config-router-af)#bgp dampening 1 500 ?
  <1-20000>  Value to start suppressing a route

R4(config-router-af)#bgp dampening 1 500 999 ?
  <1-255>  Maximum duration to suppress a stable route

R4(config-router-af)#bgp dampening 1 500 999 5
```

- **IOS XR:**

```
RP/0/RP0/CPU0:R1(config)# router bgp <ASN>
RP/0/RP0/CPU0:R1(config-bgp)# address-family ipv4 unicast
RP/0/RP0/CPU0:R1(config-bgp-af)# dampening [<half-life> <reuse-threshold> <suppress-threshold> <max-suppress-time>] [route-policy <poli
```

  - The parameters are the same. IOS XR uses `route-policy` for conditional dampening.

```
RP/0/0/CPU0:R3(config-bgp-af)#BGP Dampening ?
  <1-45>        Half-life time for the penalty
  route-policy  Route policy to specify criteria for dampening
  <cr>
RP/0/0/CPU0:R3(config-bgp-af)#BGP Dampening 5 ?
  <1-20000>  Value to start reusing a route
  <cr>
RP/0/0/CPU0:R3(config-bgp-af)#BGP Dampening 5 2 ?
  <1-20000>  Value to start suppressing a route
RP/0/0/CPU0:R3(config-bgp-af)#BGP Dampening 5 2 999 ?
  <1-255>  Maximum duration to suppress a stable route
RP/0/0/CPU0:R3(config-bgp-af)#BGP Dampening 5 2 999 10 ?
  <cr>
RP/0/0/CPU0:R3(config-bgp-af)#BGP Dampening 5 2 999 10
```

ABSTRACT

- **More Aggressive Dampening for smaller, potentially less stable prefixes** (e.g., /25 to /32): Lower suppress threshold, shorter half-life, quicker reuse.
- **Less Aggressive Dampening for larger, more stable prefixes** (e.g., /0 to /20, often aggregates from ISPs): Higher suppress threshold (allow more flaps), longer un-suppress times.

Viewing Dampened Routes

When a route is dampened, it will often be marked in the BGP table with a specific status code.

- In `show ip bgp` (IOS XE) or `show bgp ipv4 unicast` (IOS XR), a dampened route is typically marked with an `h` (history) status, indicating it's being dampened and is not currently eligible for best path.

```
*>i185.185.11.64/26   192.168.0.1              0    100      0 ?
*>i185.185.11.128/26  192.168.0.2              0    100      0 i
*>i185.185.11.192/26  192.168.0.2              0    100      0 i
 h 185.206.170.0/24   10.10.10.2                             0 65002 i
*> 185.206.171.0/24   10.10.10.2                             0 65002 i
*> 185.206.172.0/24   10.10.10.2                             0 65002 i
*> 185.206.173.0/24   10.10.10.2                             0 65002 i
*> 185.206.174.0/24   10.10.10.2                             0 65002 1618 65003 i
*> 187.187.10.0/24    10.10.10.2                             0 65002 1618 65003 i
*> 193.199.12.0/24    10.10.10.2                             0 65002 123 ?

Processed 18 prefixes, 18 paths
RP/0/0/CPU0:R3#show bgp 185.206.170.0
Wed Jun  9 04:11:30.570 UTC
BGP routing table entry for 185.206.170.0/24
Versions:
  Process           bRIB/RIB  SendTblVer
  Speaker               288         288
Last Modified: Jun  9 04:10:57.039 for 00:00:33
Paths: (1 available, no best path)
  Not advertised to any peer
  Path #1: Received by speaker 0
  Not advertised to any peer
  65002, (history entry)
    10.10.10.2 from 10.10.10.2 (10.10.2.2)
      Origin IGP, localpref 100, external
      Received Path ID 0, Local Path ID 0, version 0
      Origin-AS validity: not-found
```

Clearing BGP Dampening Information

If you want to immediately un-dampen routes (e.g., after fixing an issue causing flaps) without waiting for timers:

- **IOS XE:**

```
R1# clear ip bgp dampening [prefix/mask] // Clears dampening info for all or specific routes
```

- **IOS XR:**

```
RP/0/RP0/CPU0:R1# clear bgp [ipv4 unicast] dampening [prefix/mask]
```