Iljitsch van Beijnum

Internet Routing with **BGP**

Introduction

The internet is "a network of networks". It's made up of tens of thousands of largely independent networks, but somehow the users of one network can communicate with the users of any of the other networks. The Border Gateway Protocol (BGP) is the glue that binds these disparate networks together.

BGP is a routing protocol: its main job is to allow each network to learn which ranges of IP addresses are used where, so packets can flow along the correct route.

However, BGP has a more difficult job to do than other routing protocols. Yes, it has to make the packets reach their destination, but BGP also has to pay attention to the business side: those packets only get to flow over a network link if either the sender or the receiver pays for the privilege.

This book covers the fundamentals of the technical side of BGP, and also looks at the intersection between the technical and business aspects of internet routing.

The book contains 40 configuration examples that readers can try out on their own computer in a "BGP minilab".

Table of contents

About this book

I already wrote a book about BGP back in 2002. So why another one?

What I've learned over the years is that at its core, BGP is quite simple. However, there are many hidden nuances and caveats that people usually only begin to understand when they run into them in practice. But learning those things the hard way on a live network is less than ideal.

So what I want to do here is provide examples that are as close to real-world BGP internet routing as possible, allowing you, the reader, to start understanding the forest better by looking at some individual trees. All of this is based on running BGP training courses for almost two decades.

To keep both the writing (for me) and reading (for you) of this book manageable, the book only covers the BGP protocol and BGP configuration for connecting a network to the internet. There is a lot more to running a network, please find that information in other books and online resources. BGP is also extensively used in data centers and enterprise networks. This also not covered in this book.

You'll get the most out of this book by running the virtual example network yourself and try out the examples. With today's technology, it's possible to use Docker to run a bunch of virtual routers on a regular Windows, MacOS or Linux system. The examples are based on Free Range Routing, open source routing software that is configured very similar to "classic IOS" Cisco routers. However, the exact configuration language isn't the point; once you understand the concepts, looking up the right keywords in the vendor documentation is the easy part.

That said, if you'd like to see configuration examples for other types of routers, please let me know and I may be able to add those to a future version of this book.

Intended audience

This book is intended to be useful for anyone who wants or needs to know more about BGP, and how BGP is used for internet routing. A large part of the book discusses configuration examples, but even if you skip those, you should still get a good feel for the problems BGP solves. (And sometimes creates!)

The book is especially intended for network engineers who've just started using BGP to connect to the internet, and those who are considering doing that. Trying out the examples should give you a good feel for what that's like, and enable you to decide whether that's something you'll feel comfortable doing yourself after some study, or it's better to hire someone else to guide you through the process and then take over yourself, or perhaps outsource configuring and maintaining your BGP setup.

Internet routing

The internet consists of tens of thousands of networks that are owned and run by different companies/organizations. And yet, users of any of those networks can communicate with users of any of the other networks. It's an amazing thing.

To make this possible, at some point each network is connected to one or more other networks, creating network paths between any two locations. When two networks connect directly, routing decisions are simple: just hand off the packets to the destination network over that direct connection.

Things get more complicated when there's one or more networks in the middle that connect the source and destination network. Typically, there will be several paths that go through different intermediate networks, making routing decisions somewhat more complex. But that's nothing any routing protocol worth its salt can't handle.

However, the real complication with internet routing is that the job is not simply finding the shortest path between any two locations, but also taking into consideration the business aspects of running a network. What if networks A and B both connect to Microsoft? After all, users of both networks A and B want to be able to download their Windows updates and work on their Office365 documents with the highest possible performance.

So in theory, a user at network A can send packets to a user at network B through Microsoft. The physical connections are there, and left to their own devices, the routers will see those paths and use them if they're shorter than alternative paths.

But Microsoft is not an Internet Service Provider (ISP)—they're not in the business of providing connectivity between their users. So Microsoft will want to hide such paths in order to make sure that their

network isn't used for traffic that falls outside the scope of the services they provide.

As a result, the Border Gateway Protocol (BGP), the routing protocol that's used between the networks that collectively make up the internet, must do everything that's normally expected from a routing protocol, but in addition to that, apply policy restrictions to conform with business realities. We'll see what this means a little later in the book in the chapter Transit and peering.

The BGP protocol

Most of this chapter provides background information about BGP that doesn't directly impact operation. If you want to skip this for now, please skip ahead to the last section in this chapter, BGP operation.

"BGP" stands for "border gateway protocol". Back in 1989, when the first BGP specification was published, the word "gateway" was used for what we now call a router. So BGP really means "border router protocol". A border router is, of course, the last router in your network, which connects to the first router in the next network. BGP is the protocol these two border routers in neighboring networks use to exchange routing information.

This makes BGP an "exterior gateway protocol" (EGP), not to be confused with the exterior gateway protocol that's actually called EGP [RFC 904], which has long been obsolete. All other routing protocols are "interior gateway protocols" (IGPs), meant for handling routing within a single network. Networks that run BGP almost always also run one of the IGPs to handle their internal routing.

The IETF

Internet protocols such as BGP are developed and maintained by the Internet Engineering Task Force (IETF). The IETF is an unusual standards organization, as it doesn't have members: everyone can participate simply by joining the mailing lists for the different working groups. Three times a year, there are IETF meetings. The meeting fee (currently $875) is the main source of revenue for the IETF. As there is no formal participation, IETF decision making is done by "rough consensus". This means a decision must be supported by a large majority of those who express an opinion, but it doesn't have to be completely unanimous.

IETF standards and other documents are published as a "request for comment" (RFC). Each RFC has a number. A new version of a docu-

ment is published under a new RFC number. RFCs start their life as a working group "internet-draft", which is iterated until the document is ready for publication as an RFC. Individuals may also write drafts, which may or may not be adopted by a working group and progress to an RFC.

Not every RFC is a standard. RFCs that specify a protocol that is intended to become an official standard at some point are published as "standards track". Within standards track, a document used to start as a "draft standard". That stage is now merged with "proposed standard". After significant operational experience and refinement, a protocol specification may become an official internet standard.

BGP-4 [RFC 4271] is still a draft standard—moving protocols along through the standards track process isn't always given the highest priority within the IETF.

Protocol specifications may also be published as "experimental". Documents of various kinds are published as "informational" and operational guidance may become "best current practice" and receive a BCP number. When a document is no longer relevant it is given the status "historic".

The best way to read RFCs online is as the HTML version at the RFC Editor website www.rfc-editor.org. Originally, RFCs were published in a very simple text-only format. The HTML versions add information about a document's status at the top, as well as links to related RFCs.

Distance vector vs link state

And now it's time for some routing protocol theory. There are two ways to distribute routing information through a network: distance vector and link state. The idea behind distance vector is that a router collects routing information (paths towards each prefix) from its neighbors, then chooses the best path towards each prefix, and tells its neighbors that best path. Alternative paths that are not considered "best" at this time thus remain hidden from other routers.

With link state protocols, a router doesn't tell its neighbors about the *conclusions* of its path calculations, but rather, the *data* it used to reach those conclusion. So each router independently calculates the best path to reach each destination.

Link state protocols have the advantage that they're faster than distance vector protocols. With a link state protocol, whenever a router detects that it has lost the connection to a neighboring router, it will send out an update to its remaining neighbors, which is quickly "flooded" throughout the network. Then each router recalculates the best paths. With a distance vector protocol, a router first has to recompute all paths, and only then it can inform its remaining neighbors of the change.

A limitation of link state protocols is that all routers must use the same algorithm and the same parameters to calculate paths. If they didn't, routing loops would be possible.

The main example of a distance vector protocol is RIP [W]. RIP is a very simple protocol that uses a hop count as a way to determine which path is best. That can mean that one 1 Gbps hop is preferred over two 10 Gbps hops, which is usually not what you'd want. A big downside of RIP is that it's very slow to react to lost connectivity due to the count-to-infinity problem [W]. The current IPv4 version of RIP is RIPv2, the IPv6 version is RIPng.

Cisco built its own more advanced distance vector routing protocols: IGRP and EIGRP [W].

OSPF is the most widely used example of a distance vector protocol. With OSPF, each link between two routers has a "cost" associated with it, and OSPF then uses the "Dijkstra" a.k.a. "shortest path first" (SPF) algorithm to calculate the best path between any two points in the network. The current version for IPv4 is OSPFv2 `[RFC 2328]` and for IPv6 OSPFv3 `[RFC 5340]`.

IS-IS [W] is a link state protocol created for the OSI CNLP protocol. It was later extended to also support routing IPv4 and IPv6. IS-IS is mainly used in very large IP networks.

Which brings us to BGP: is it a distance vector or a link state protocol?

As we'll discuss in the chapter Transit and peering, internet routing requires using policies that limit the propagation of routing information. This makes it impossible to use a link state routing protocol for inter-domain routing. So BGP is *mostly* a distance vector protocol, but unlike other distance vector protocols, BGP carries path information in its updates. This makes it possible to detect routing loops much faster, so BGP can reroute more quickly after a failure than a simple distance vector routing protocol such as RIP.

BGP versions

BGP version 1 was published in 1989 [RFC 1105]. Versions 2 [RFC 1163] and 3 [RFC 1267] quickly followed over the next two years. With version 3, BGP looked a lot like the BGP we know today, except that it still only supported classful addressing. BGP-4 added support for classless inter-domain routing. BGP-4 was first published in 1994 [RFC 1654]. There have been two revisions of the specification (not of the protocol), with the most recent one published in 2006 [RFC 4271].

Amazingly, we still use BGP version 4 today, 28 years after the protocol specification was first published. There are two reasons for this:

1. It's really hard to change the routing protocol that's used internet-wide.
2. BGP-4 is designed to be extended in backward compatible ways, so new features could be added without having to create a new version of the protocol.

Autonomous Systems

Networks that run BGP are called *autonomous systems* (ASes). The idea is that each AS presents a consistent view of itself to the outside world, and what happens inside an AS is irrelevant to other ASes, as far as BGP is concerned.

One definition of an AS as "all routers under common administrative control". However, that definition doesn't work for service provider networks, as the service provider only has administrative control over its own routers; many customers administer their own routers themselves. But if these customer routers don't run BGP themselves, they're still part of the service provider's AS.

Each AS has an AS number. These used to be 16-bit numbers, but BGP was extended to support 32-bit (sometimes called "4-byte" or "4-octet") AS numbers. As of the middle of the 2010s, all BGP routers support 32-bit AS numbers. But if a router doesn't understand 32-bit AS numbers, it will simply see AS number 23456 any time an AS number shows up that's not 16-bit compatible.

BGP neighbor relationships

Like all routing protocols, BGP maintains relationships with neighboring routers. *Unlike* other routing protocols, BGP doesn't discover neighboring routers automatically. Instead, BGP neighbor relationships must be explicitly set up on both sides through administrative configuration. I.e., you'll have to tell the router the IP addresses of its neighbors along with the remote AS number and other information that's relevant to that specific neighbor relationship. We'll start doing that in the chapter BGP configuration 101.

BGP routers communicate with their neighbors over TCP port 179. Both neighbors try to connect to the other on port 179. This means that sometimes router A is the "client" and router B is the "server", and sometimes the other way around. After the TCP session has been established, the two routers start to exchange BGP messages. The TCP

session stays connected indefinitely. So it's not unusual to see BGP TCP sessions that have been up for weeks or even months.

When the TCP session goes away, the BGP routers on both sides throw out all the routing information they've learned over that BGP session and then try to set up a new TCP session.

BGP messages

When a BGP TCP session connects, the two routers will start to exchange BGP messages. The following is a brief description of each message type; for detailed information see section 4 of RFC 4271.

All BGP messages start with a "marker" for compatibility with older BGP versions, with the rest of the message following the type-length-value model [W]. There are five BGP messages:

1. Open
2. Update
3. Keepalive
4. Notification
5. Route-refresh

The **Open message** contains a version field, which was useful during the transition from BGP-3 to BGP-4. The router also puts its AS number, its router ID and its configured hold time in the Open message. The router ID is a 32-bit value that's unique for a router (usually one of its IPv4 addresses) and the hold time is how long the router will wait before declaring the BGP session dead when it doesn't see any incoming BGP messages.

Last but not least, there's room for optional parameters. These are typically used to negotiate the use of BGP extensions.

The **Update message** does most of BGP's heavy lifting. An Update message can carry withdrawn routes, new routes or both. Any withdrawn routes simply go in the "withdrawn routes" field. New routes,

if they're included in the update, use two fields: path attributes and NLRI.

The withdrawn routes are routes (prefixes) that the neighbor had previously told us we could reach through them, but now this is no longer the case. So the local router removes those paths from its BGP table. See the section BGP operation later this chapter for how this works.

Path attributes are different kinds of information that BGP associates with each prefix. The two most important ones are the AS path, which shows all the ASes between the local router and the destination prefix, and the next hop address, which is the address we have to send packets to in order for those packets to reach the destination in question.

NLRI stands for network layer reachability information, which is just a fancy way of saying "one or more IP prefixes". There's only one set of path attributes, so if the NLRI field contains multiple prefixes, those all have the same path attributes. Prefixes with different path attributes are transmitted in separate Update messages.

The **Keepalive message** contains no information: it just has the fixed marker, the type is 3, indicating a Keepalive message, and the length is zero. Keepalive messages are sent periodically in order to make sure that the neighbor sees we're still alive and thus the session's hold timer at the neighbor's side doesn't reach zero. See the Making BGP faster chapter for more information.

Routers send a **Notification message** when they need to tear down the BGP session. This is usually because an error has occurred, but also when the session needs to be terminated because of maintenance, or as part of capabilities negotiation. The Notification message has an error code and an error subcode as well as room for optional additional data.

The **Route-refresh message** is an addition to BGP [RFC 2918] to allow a router to ask a neighbor to send all BGP updates again. This way, new filters can be applied to those updates. See the chapter Filtering BGP for more information.

Path attributes

There are four types of path attributes:

1. Well-known mandatory: all prefixes must carry this path attribute.
2. Well-known discretionary: all BGP implementations must be able to process this path attribute, but prefixes may or may not carry this attribute.
3. Optional transitive: BGP implementations aren't required to process these. If a router encounters an optional transitive path attribute that it doesn't understand, it has to propagate the attribute to its neighbors unchanged.
4. Optional non-transitive: BGP implementations aren't required to process these. If a router encounters an optional non-transitive path attribute that it doesn't understand, it removes the attribute.

IANA is the organization that keeps track of internet-related protocol numbers. The IANA BGP attributes registry currently lists nearly 40 path attributes. These are the ones defined in the BGP specification:

1. `ORIGIN` (well-known mandatory): indicates whether a path was learned from an IGP, from the EGP protocol or is "incomplete", meaning it was learned through some other means. The `ORIGIN` attribute doesn't seem to perform any function.
2. `AS_PATH` (well-known mandatory): the list of ASes that have "seen" this path. Used for loop suppression and may also be used for filtering and policy.
3. `NEXT_HOP` (well-known mandatory): the address of the next hop router, which is normally the address of the BGP neighbor that sent the update.

4. `MULTI_EXIT_DISC` (optional non-transitive): the multi exit discriminator (MED) is also often called "metric". Is used to choose between paths learned from the same neighboring AS.

5. `LOCAL_PREF` (well-known mandatory): the local preference carries a path's degree of preference. This attribute must be present on updates within an AS (iBGP), but not on updates that go to external ASes (eBGP).

6. `ATOMIC_AGGREGATE` (well-known discretionary): used when routers perform aggregation. This was relevant in the transition from BGP-3 to BGP-4, but is rarely used today.

7. `AGGREGATOR` (optional transitive): also used for aggregation.

The following are path attributes that were added later to BGP, and are thus optional.

- `COMMUNITY` (transitive, `[RFC 1997]`): carries one or more 32-bit labels that can be used for various purposes. See the Filtering BGP and Traffic engineering chapters for more information.

- `ORIGINATOR_ID` and `CLUSTER_LIST` (non-transitive, `[RFC 4456]`): used by BGP route reflectors, see the chapter iBGP.

- `MP_REACH_NLRI` and `MP_UNREACH_NLRI` (non-transitive, `[RFC 4760]`): carry multiprotocol extensions, see the section Multiprotocol BGP later this chapter.

- `EXTENDED COMMUNITIES` (transitive, `[RFC 4360]`): supports larger communities of different types. Not very widely used (for internet routing) because each of the different types of extended communities needs to be supported explicitly by a BGP implementation.

- `AS4_PATH` (transitive, `[RFC 6793]`): carries the 32-bit version of the AS path. 32-bit capable routers update both the `AS4_PATH` as well as `AS_PATH`, inserting "23456" as a placeholder for 32-bit AS numbers. 16-bit capable routers of course only update the `AS_`–

PATH, but the next 32-bit router will add any AS hops missing from the AS4_PATH using the AS_PATH.

- LARGE_COMMUNITY (transitive, [RFC 8092]): larger communities.
- BGPsec_Path (non-transitive, [RFC 8205]): path attribute that carries the protected AS path as per the BGPsec security mechanism. See the section on BGPsec in the BGP security chapter.

Multiprotocol BGP

The routing protocols we still use today were all initially created in the 1980s, long before IPv6 saw the light of day. Of course, once IPv6 arrived, it also needed routing protocols. For RIP and OSPF, new versions of those protocols were built from the ground up. This is the "ships in the night" concept: RIPv2 and OSPFv2 handle IPv4 routing while RIPng and OSPFv3 handle IPv6 routing. Other than their basic design, the IPv4 and IPv6 versions of these routing protocols are completely separate and they don't interact at all.

IS-IS uses the opposite approach: the one IS-IS protocol handles IPv4 and/or IPv6 routing alongside the OSI CLNP for which it was created.

Like IS-IS and unlike RIP and OSPF, there's just one BGP that handles both IPv4 and IPv6. This is made possible by the BGP multiprotocol extensions [RFC 4760].

Rather than just add support for IPv6, multiprotocol BGP adopts the "address family" concept, with IPv4 and IPv6 being different address families, along with other address families such as Ethernet VPN (EVPN).

A set of related protocols, such as TCP/IP is called a protocol stack or a protocol family. At some point, the idea was that a protocol family like TCP/IP would support multiple address families, but that never worked out, so in practice there is no difference between a protocol family and an address family. The term "address family" is the one used with multiprotocol BGP, although "protocol family" would probably be clearer.

In BGP, address families are identified with the address family identifier (AFI). There's also a subsequent address family identifier (SAFI) that's used to differentiate between (for instance) prefixes used for unicast (one-to-one) and multicast (one-to-many) communication. IANA maintains AFI and SAFI registries.

Multiprotocol BGP is implemented through two new path attributes already mentioned earlier this chapter: `MP_REACH_NLRI` and `MP_UNREACH_NLRI`. The `MP_UNREACH_NLRI` replaces the "withdrawn routes" field in the BGP Update message, containing an AFI and SAFI and then NLRI formatted in the way specified for that AFI and SAFI. For instance, IPv4 NLRI is encoded as a one-byte length field that holds the prefix length and a variable length prefix field.

So for instance a `/20` prefix would be three bytes in length that hold the 20-bit prefix padded to 24 bits to make the prefix value three bytes long. Interestingly, the RFC that describes the use of the multiprotocol extensions for IPv6 `[RFC 2545]` doesn't even bother specifying the same for IPv6. Obviously the only difference is that the prefix length can now be up to 128 rather than 32.

The `MP_REACH_NLRI` attribute replaces the NLRI field in the Update message. Like `MP_UNREACH_NLRI` it holds an AFI, SAFI and NLRI, but in addition to those fields, also a next hop length field and a variable length next hop field.

All interfaces that have IPv6 enabled must have a link local address in addition to any regular global unicast addresses. Link local addresses are addresses that are only used locally on a subnet, and thus don't have to be globally unique. They fall within the prefix `fe80::/64`. Routes in routing tables typically point to the link-local addresses of routers.

Because IPv6 requires routing protocols to carry link local addresses, when using the IPv6 AFI, multiprotocol BGP carries a link local next hop address as well as a global next hop address where appropriate. When only a global next hop address is present, the next hop length is

16 (128 bits), when there's also a link-local next hop address, it's 32 (2 × 128 bits).

`MP_REACH_NLRI` and `MP_UNREACH_NLRI` replace the withdrawn routes and NLRI fields in the Update message, so these remain empty in multiprotocol BGP operation. Other path attributes are included as usual.

With multiprotocol BGP, it's possible to run the BGP TCP session either over IPv4 or over IPv6, and the session can carry IPv4 and/or IPv6 prefixes. Routers will announce the AFIs/SAFIs they want to enable on a new session in the Open message. To avoid problems with next hop address processing, it's best to use an IPv4 BGP session to exchange IPv4 prefixes with neighboring ASes and an IPv6 BGP session for IPv6 prefixes. For iBGP this is slightly different, as we'll see in the iBGP chapter.

BGP states and finite-state machine

A BGP session can be in one of six *states*. The relationship between these states and the 28 events that can move the session from one state to another are modeled using a finite-state machine (FSM) [W]. The BGP RFC describes the FSM in detail; this is a simplified version:

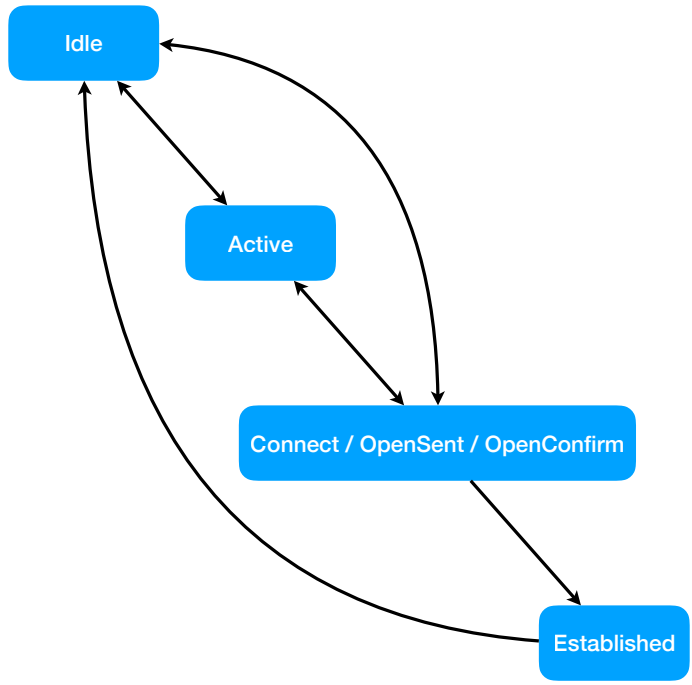

Figure 1. A simplified version of the BGP finite-state machine

BGP sessions start in the **Idle** state. In the Idle state, the router doesn't try to connect to the neighbor in question, and incoming connection attempts are rejected. It is possible to move directly from the Idle to the Connect state, but usually, when the router is ready to start a BGP session, the session first moves to Active.

In the **Active** state, there is no active connection yet, but the router *actively* tries to connect to its neighbor. From Active, the connection can move to Connect, OpenSent or OpenConfirm.

Usually, a BGP session progresses quickly through the **Connect**, **OpenSent** and **OpenConfirm** states, so in figure 1 above those states are collapsed into a single one. Upon various error conditions, a connection may revert back to Idle or Active. But if everything goes according to plan, the session moves into the Established state.

In the **Established** state, the two routers on opposite sides of the BGP session are ready to exchange routing information in the form of BGP

Update messages. It may take some time for the initial set of updates to be exchanged after a session enters the Established state. If an error occurs the session returns to the Idle state.

BGP operation

In this section, we'll have a look at how BGP routers exchange prefixes. An important rule is that a router may only propagate (announce) to its neighbors paths that it actually uses itself. So if a router has a choice of multiple paths towards a given destination prefix, it must first select the best one out of these paths.

BGP best path selection is somewhat complex, and we'll discuss it in more detail in the Traffic engineering chapter. For now, we'll just look at the AS path length, and consider the path with the smallest number of AS hops in the AS path best.

We'll look at the flow of BGP updates between two autonomous systems, AS 10 to the left and AS 40 to the right. At this point, AS 10 and AS 40 don't have a BGP session established between them yet:

```
          AS 10                          AS 40
  Network        Path              Network        Path
> 192.0.2.0      20 30 82       > 192.0.2.0      82
> 198.51.100.0   4206           > 198.51.100.0   4206
```

Both ASes have two prefixes in their BGP table: the `192.0.2.0/24` and the `198.51.100.0/24` prefixes. (The `/24` prefix length is implied for these class C networks.) AS 10 can reach the 192-prefix through two intermediate hops and is directly connected to AS 4206, the origin of the 198-prefix. For AS 40, both prefixes are reachable directly over one-hop paths.

Assuming no filters, when the BGP session between AS 10 and AS 40 establishes, they each send a copy of their full BGP table to their neighbor:

```
          AS 10                              AS 40
  Network          Path                 Network          Path
> 192.0.2.0        40 82       <=     > 192.0.2.0        82
                   20 30 82     =>                        10 20 30 82
  198.51.100.0     40 4206     <=     > 198.51.100.0     4206
>                  4206         =>                        10 4206
```

So at this point, both ASes now have two paths towards each prefix: the one they already had, and the new one from the other AS. By sending each other copies of these prefixes, the routers in both ASes invite the other to send traffic to these destinations through them.

What we see, as indicated by the > character, is that the AS 10 router takes AS 40 up on its offer to send traffic to the 192-prefix through AS 40, as that path is two hops (`40 82`) while the path AS 10 already had is three hops (`20 30 82`). (When a router propagates a path, it adds its own AS number to the left of the existing AS path.)

However… moments earlier the AS 10 router had invited AS 40 to send traffic to the 192-network through AS 10. Should AS 40 want take AS 10 up on that offer, we'd be in the situation where AS 10 tries to reach `192.0.2.0/24` through AS 40, while AS 40 tries to reach `192.0.2.0/24` through AS 10. This means we have a routing loop on our hands and packets will pingpong between AS 10 and AS 40.

So in order to prevent this eventuality, the AS 10 router sends an Update message to the AS 40 router withdrawing the 192-prefix. When AS 40 has processed this update and removed from its BGP table the path towards the 192-prefix through AS 10, BGP has reached a stable state:

```
          AS 10                              AS 40
  Network          Path                 Network          Path
> 192.0.2.0        40 82               > 192.0.2.0        82
                   20 30 82 => x
  198.51.100.0     40 4206             > 198.51.100.0     4206
>                  4206                                   10 4206
```

Note that for the 198-network, each router keeps the new path in its BGP table. For both, their original path is shorter and therefore preferred over the new path learned from the other AS. So in this case,

there is no conflict. Should one of the routers decide to start using the path through the other, it will send a withdraw at that point.

In this stable state, no further updates are sent, just periodic Keepalive messages to make sure the BGP session is still operational. When a router stops receiving Keepalive messages or loses the BGP TCP session, it removes all paths learned from the neighbor from its BGP table, selecting new best paths as necessary, and starts trying to re-establish the BGP session. When it does, prefixes are exchanged again as described above.

BGP configuration 101

With all the preparations out of the way, we're now ready to start configuring a router to speak BGP!

The assumption is that the router is already set up, has connectivity to two ISPs, and that the router interfaces towards those ISPs are configured with the right IP addresses. Example 1 shows the simplest possible BGP configuration with two ISPs.

Example 1: A very simple BGP configuration

```
!
router bgp 65082
 network 192.0.2.0/24
 neighbor 192.0.2.21 remote-as 65030
 neighbor 192.0.2.21 description ISP 30
 neighbor 192.0.2.41 remote-as 65040
 neighbor 192.0.2.41 description ISP 40
!
```

If you want to try this example and the other examples for yourself, have a look at Installing the minilab and running examples at the end of the book. If you've never configured a router using a Cisco-like command line interface (CLI), have a look at Appendix: the router CLI for a short introduction.

To avoid issues with other examples and to keep consistency between the examples, the addresses for both BGP neighbors (`192.0.2.21` and `192.0.2.41`) fall within out own prefix `192.0.2.0/24`. In reality, ISPs normally provide a /30 or /29 prefix to number the link subnet between the ISP and the customer.

The `router bgp 65082` line tells the router that we want to configure the BGP protocol, and that this router belongs to AS 65082. The next line tells the router that we want to *originate* the prefix `192.0.2.0/24`. Originate means that this router injects this prefix into BGP and tells the rest of the world that these addresses are used in our AS.

On Cisco routers, we can't specify our prefix or prefixes using CIDR notation. Instead, we'll have to use a mask. In this case that would be `network 192.0.2.0 mask 255.255.255.0`. But when displaying the configuration, the mask part will be left out, as the mask that corresponds to `/24` is implied for class C networks. With the FRRouting software for Linux, either prefix notation or a mask is accepted.

We can monitor the progress of the BGP session establishment with the `show ip bgp summary` command. This is what an older router would show if we asked it what's going on with BGP:

```
Router# show ip bgp summary
BGP router identifier 192.0.2.251, local AS number 65082
RIB entries 1, using 112 bytes of memory
Peers 2, using 40 KiB of memory

Neighbor        AS MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down   State
192.0.2.21   65030      81      81       6    0    0 00:00:04       2
192.0.2.41   65040       0       0       0    0    0 never    Active
```

This will look a little different when you try the example yourself, as the output of the router commands sometimes has to be edited so the lines don't get too long and some less relevant information is left out. Also, different routers will show slightly different output, but they will largely show the same information.

For the first neighbor, the state is a number. This means the BGP session is in the Established state, and the number is the number of prefixes received and accepted from the neighbor. (I.e., prefixes filtered out don’t count.)

Should the InQ or OutQ numbers be higher than zero, this means the routers are still busy exchanging prefixes. However, a zero here doesn’t necessarily mean they’re *not* exchanging prefixes currently.

The second neighbor is in the Active state, and has never been up (in the Established state). If this persists or if the state goes to Idle, there’s likely a problem that warrants talking to someone who can check the

other end of the BGP session. But in the case above we were just a bit impatient and the second BGP session came up a few moments later.

What happened in this example is that immediately after we enter the `neighbor ... remote-as ...` line, the router started trying to set up a BGP session to the specified neighbor. Even before we set up filters or other restrictions. This way, our AS will happily propagate the information it learns from AS 65030 to AS 65040 and vice versa.

That is not good. So some newer routers will not send any outgoing updates until an outgoing filter or policy is configured and not accept incoming updates until an incoming filter or policy is configured, as per [RFC 8212]. So FRRouting version 8 (which is used if you want to run the examples yourself using the Docker BGP minilab), you'll get the following results with the example 1 configuration in effect:

```
Router# show ip bgp summary

IPv4 Unicast Summary (VRF default):
BGP router identifier 192.0.2.251, local AS number 65082 vrf-id 0
BGP table version 1
RIB entries 1, using 192 bytes of memory
Peers 2, using 1433 KiB of memory

Neighbor        AS MsgRcvd MsgSent   Up/Down State/PfxRcd   PfxSnt
192.0.2.21   65030      20      16  00:13:41     (Policy) (Policy)
192.0.2.41   65040      18      16  00:13:41     (Policy) (Policy)
```

For the moment, let's work around that by entering:

```
Router# conf t
Router(config)# router bgp 65082
Router(config-router)# no bgp ebgp-requires-policy
Router(config-router)# exit
Router(config)# exit
Router# clear ip bgp *
```

So first we add `no bgp ebgp-requires-policy` to the configuration, and then issue the `clear ip bgp *` command to restart all the BGP sessions so we can be sure that we're not looking at stale information. We now get:

```
Router# show ip bgp summary

IPv4 Unicast Summary (VRF default):
BGP router identifier 192.0.2.251, local AS number 65082 vrf-id 0
BGP table version 4
RIB entries 9, using 1728 bytes of memory
Peers 2, using 1433 KiB of memory

Neighbor        AS MsgRcvd MsgSent   Up/Down State/PfxRcd PfxSnt
192.0.2.21   65030      16      12  00:00:05            2       4
192.0.2.41   65040      14      14  00:00:05            2       4
```

The fact that the router sends four prefixes to each neighbor is a bit unexpected. So let's see which prefixes it's sending to neighbor `192.0.2.21`:

```
Router# show ip bgp neighbors 192.0.2.21 advertised-routes
Status codes:  s suppressed, d damped, h history, * valid, > best,
= multipath,
               i internal, r RIB-failure, S Stale, R Removed
Nexthop codes: @NNN nexthop's vrf id, < announce-nh-self
Origin codes:  i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found

   Network       Next Hop Metric LocPrf Weight Path
*> 10.0.10.0/23  0.0.0.0                  0 65040 65010 i
*> 10.0.20.0/22  0.0.0.0                  0 65030 65020 i
*> 10.0.30.0/23  0.0.0.0                  0 65030 i
*> 10.0.40.0/21  0.0.0.0                  0 65040 i

Total number of prefixes 4
```

It's a bit odd that FRRouting sends prefixes it just learned from AS 65030 back to AS 65030, but that shouldn't cause problems. And indeed it sends the AS 65040 prefixes to AS 65030 (as well as the other way around), so in the next chapter we're going to add some filters to keep that from happening.

However, our own prefix `192.0.2.0/24` is *not* advertised to this neighbor. The reason for that is simple:

```
Router# show ip route 192.0.2.0/24
% Network not in table
```

So our own prefix is not in our router's IP routing table. In that situation, the logic is that if the router itself doesn't know where to send packets for this prefix, how can it advertise this prefix to the rest of the world? We can fix this using a static route:

Example 2: a static route to enable prefix origination

```
!
ip route 192.0.2.0 255.255.255.0 Null0 250
!
```

The `Null0` interface is a special interface that makes packets forwarded to it disappear. The effect of this static route is that packets towards `192.0.2.x` are filtered out. Using a Null0 route like this has the added benefit that if parts of the prefix in question aren't in use, packets won't be sent back to the ISP if there's a default route, with the ISP then sending the packets back and they keep ping ponging back and forth until their time to live reaches zero.

The 250 is the priority of the static route. Any other routes for that same prefix with a lower priority value will override the Null0 route. With this route in effect, the router advertises the prefix to its neighbors:

```
Router# show ip bgp 192.0.2.0/24
BGP routing table entry for 192.0.2.0/24, version 6
Paths: (1 available, best #1, table Default-IP-Routing-Table)
  Advertised to non peer-group peers:
  192.0.2.21 192.0.2.41
  Local
    0.0.0.0 from 0.0.0.0 (192.0.2.255)
      Origin IGP, metric 0, localpref 100, weight 32768, valid,
sourced, local, best
```

Of course no BGP configuration is complete without some IPv6. Example 3 below is the IPv6 equivalent of examples 1 and 2, except that we're only configuring an IPv6 BGP session towards ISP 30 and not ISP 40.

Example 3: The IPv6 version of examples 1 and 2

```
!
router bgp 65082
 neighbor 2001:db8:30:8201::1 remote-as 65030
 neighbor 2001:db8:30:8201::1 description ISP 30
 no neighbor 2001:db8:30:8201::1 activate
!
 address-family ipv6
 network 2001:db8:82::/48
 neighbor 2001:db8:30:8201::1 activate
 exit-address-family
!
ipv6 route 2001:db8:82::/48 Null0 250
!
```

The obvious difference is that the neighbor address is an IPv6 address. However, even for IPv6 neighbors, the assumption is that we're going to exchange IPv4 prefixes, not IPv6 prefixes. So what we do in the first part of the configuration, is disable IPv4 for this BGP session with the `no neighbor ... activate` command.

Next, we tell the router we want to configure parameters related to address family IPv6. (We could have said `address-family ipv6 unicast` to be more precise.) Here we can specify the IPv6 prefix(es) we want to originate, `2001:db8:82::/48` in this case. Last but not least, we activate our neighbor for the IPv6 unicast address family. And we add the static route to the `Null0` interface. The results, using the slightly reordered `show bgp ipv6 ...` vs `show ip bgp ...`:

```
Router# show bgp ipv6 unicast summary
IPv6 Unicast Summary (VRF default):
BGP router identifier 192.0.2.251, local AS number 65082 vrf-id 0
BGP table version 2
RIB entries 3, using 576 bytes of memory
Peers 1, using 716 KiB of memory

Neighbor                AS   Up/Down State/PfxRcd PfxSnt Desc
2001:db8:30:8201::1  65030  00:03:11            1      2 ISP 30
```

And:

```
Router# show bgp ipv6 unicast
BGP table version is 2, local router ID is 192.0.2.251, vrf id 0
Default local pref 100, local AS 65082
Status codes:  s suppressed, d damped, h history, * valid, > best,
= multipath,
               i internal, r RIB-failure, S Stale, R Removed
Nexthop codes: @NNN nexthop's vrf id, < announce-nh-self
Origin codes:  i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found

   Network          Next Hop        Metric LocPrf Weight Path
*> 2001:db8:30::/44 fe80::42:acff:fe11:4
                                         0             0 65030 i
*> 2001:db8:82::/48 ::                   0         32768 i

Displayed  2 routes and 2 total paths
```

The `fe80::` next hop address is an IPv6 link local address. All IPv6 routing protocols are required to use link local next hop addresses, as link local addresses are required so routers can send ICMPv6 redirect [RFC 4861] messages when necessary. However, for iBGP to work properly, regular global unicast next hop addresses are required. Which is also present if we further inspect the prefix in question:

```
Router# show bgp ipv6 unicast 2001:db8:30::/44
BGP routing table entry for 2001:db8:30::/44, version 2
Paths: (1 available, best #1, table default)
  Advertised to non peer-group peers:
  2001:db8:30:8201::1
  65030
    2001:db8:30:8201::1 from 2001:db8:30:8201::1 (198.51.100.223)
    (fe80::42:acff:fe11:4) (used)
      Origin IGP, metric 0, valid, external, best (First path
received)
```

Appendix: BGP minilab

You can run most of the examples in a "BGP minilab" so you can see how they work and perform your own experiments. The minilab uses virtual FRRouting routers that run in Docker containers. The practice network is set up as follows:

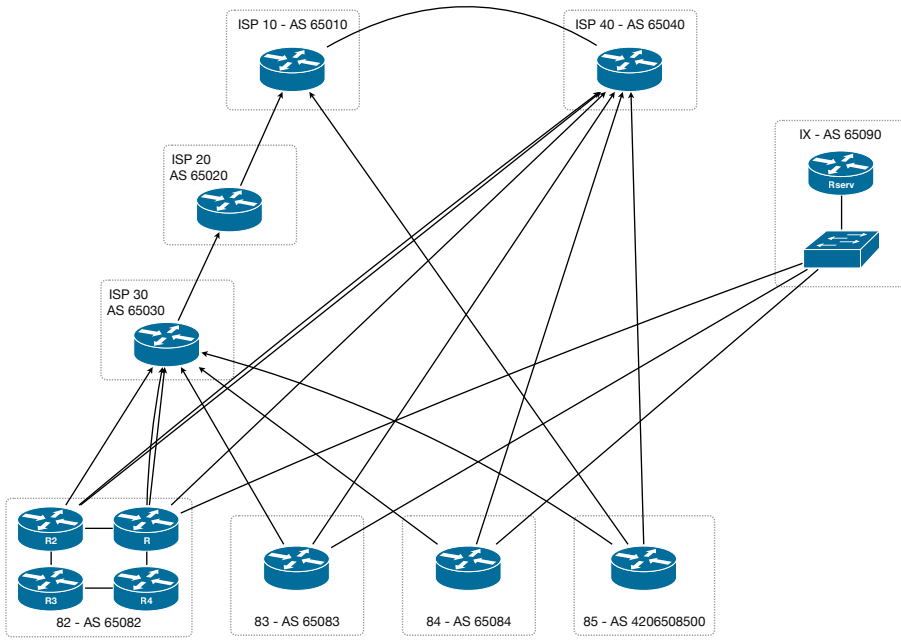

Figure 8: BGP mini lab practice network

The components of the practice network are:

- "Network 82", our own network. The main router is R1 or simply Router, with three additional routers (R2, R3 and R4) that are used in later examples. Network 82 gets transit service from ISPs 30 and 40, and can peer with networks 83, 84 and 85 through the internet exchange.
- ISPs 10, 20, 30 and 40, where ISPs 10 and 40 sit at the top of the hierarchy and peer with each other. ISP 30 is a transit customer of ISP 20, and ISP 20 is a transit customer of ISP 10.

- An internet exchange with a route server.
- Three peers: networks 83, 84 and 85. These are also all customers of ISPs 30 and 40 and connect to the internet exchange.

Installing the minilab and running examples

Install the minilab on your own computer as follows:

- Install Docker
- Under Windows: make sure it's possible to run Powerhell scripts
- Download the example configurations and the supporting scripts from my website and unzip them
- Start Docker
- Start the command line: terminal (Mac), shell or xterm (Linux) or Powershell (Windows) and make the folder/directory with the downloaded examples your current directory

With Docker running you can use the following scripts:

- `start.sh` / `start.ps1`: starts the virtual routers and loads the configurations to run an example
- `connectrouter.sh` / `connectrouter.ps1`: connects to an already running virtual router
- `stoprouters.sh` / `stoprouters.ps1`: stops all running virtual routers

To run an example, use the example script followed by the example number (or name). So on Mac/Linux:

```
./start.sh example 1
```

On Windows:

```
.\start.ps1 example 1
```

This will start up the required virtual routers and connect you to the main router “Router” a.k.a. Router82. When you log out, all the virtual

routers are shut down. If you want to run several examples, add the `keeprunning` argument when starting an example, like:

```
.\start.ps1 keeprunning 1
```

This way, when you disconnect from the virtual router, some of the "support" virtual routers are kept running so they don't have to be restarted when starting another example. You can use the additional keyword `detach` to run router82 in the background, making it easier to connect to different routers. Some examples do this automatically.

The `connectrouter` scripts take a router number as the first argument and will connect you to that router. When you log out, the virtual router keeps running. You can also add a command and then the script will run that command then return while the virtual router keeps running:

```
% ./connectrouter.sh 82 show ip bgp summary
Router# show ip bgp summary
BGP router identifier 192.0.2.251, local AS number 65082
RIB entries 5, using 560 bytes of memory
Peers 2, using 18 KiB of memory

Neighbor        AS MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down  State
192.0.2.21   65030       3       6       0    0    1 00:00:02     1
192.0.2.41   65040       4       5       0    0    1 00:00:02     1

Total number of neighbors 2

Total num. Established sessions 2
Total num. of routes received     2
%
```

Use the `stoprouters` script to stop all running virtual routers.

Saving your configuration overwrites the existing example configuration. So use the `write` command with care.

About the author

Iljitsch van Beijnum got his start in the Dutch Internet Service Provider business in 1995. He soon realized that in order to maintain more than one connection to the internet, you need something called "BGP". In 1997, he co-founded Pine Internet (later Pine Digital Security). In 1999, he worked for UUNET Netherlands on designing and implementing a new Dutch high speed backbone.

In 2000, Iljitsch started his own business now called inet6 consult. Between 2000 and 2007, he mostly did work for web hosting companies, among other things helping them connect to internet exchanges.

In 2002, he authored "BGP, Building Reliable Networks with the Border Gateway Protocol", published by O'Reilly, and in 2005 "Running IPv6", published by Apress. He also started attending IETF meetings in 2002.

In 2007, Iljitsch started writing for Ars Technica. Later that year he moved to Spain to become a research assistant at UC3M, where he did more IETF work, most notably on NAT64 and DNS64 as well as a suggested improvement to BGP. Iljitsch holds a bachelor's degree in Information and Communication Technology from the Haagse Hogeschool in The Hague and a master's degree in telematics from UC3M Madrid.

After returning to the Netherlands, in 2016 Iljitsch joined Logius, an agency of the Dutch Ministry of the Interior and Kingdom Relations. As a network architect, he was responsible for the Dutch government-wide IPv6 numbering plan. In 2019 he left Logius to return to being independent.

Follow Iljitsch on Twitter or connect on LinkedIn.

Table of contents of the full book