Chapter 27

Border Gateway Protocol (BGP) and Attacks

Contents

27.1 Introduction

Border Gateway Protocol (BGP) is the standard exterior gateway protocol designed to exchange routing and reachability information among autonomous systems (AS) on the Internet. It is the “glue” of the Internet, a critical piece of the Internet infrastructure, and essential for understanding how the Internet works. Due to its importance, BGP is a primary attack target. If attackers can compromise BGP, they can disconnect the Internet and redirect traffic. This chapter focuses on explaining how the BGP protocol works in practice, and how it can be attacked

Every year before teaching BGP in my class, I would run traceroute from my home, just to see how my packets travel to different locations. I was amazed by what I saw. For many years, my packets to the Syracuse University had to go to Philadelphia, and then came back. I live only 10 miles away from the university. This changed a few years ago. Now my packets to the university no longer take such a huge detour.

There are many other surprises. For example, when I traceroute to two different organizations, and I know they are physically located near each other, but the route taken by my packets are very different. Some companies are quite far from where I live, but from the traceroute, it seems that they are just nearby.

Looking at the Internet from the sky To understand the reasons behind these interesting observations, we need to understand how the Internet is formed. Let us look at the Internet from the sky. At a high altitude, we will see that the Internet consists of many networks connected by routers. Packets go from one network to another via the routers, until they reach the destination.

Let us descend a little bit, and look at the Internet from a lower altitude. We start to see that these networks are not completely independent. Some networks belong to the same organization. Packets actually stays inside one organization as long as they can, but eventually get switched to another organization. The transition typically happens at a data center or a facility called Internet exchange.

Our journey inside a packet. Finally, we have descended to the ground. Let us hop on a magic school bus (i.e., a packet), sit inside its payload area, buckle up the seat belt, and start our journey towards another university. This time, we will travel far, to the university where I did my undergraduate studies, the University of Science and Technology of China (USTC). Our journal starts from Syracuse, New York.

In the first leg of our journal, we stayed inside the networks belonging to `Spectrum.com`, `rr.com`, and `charter.com`, which all belong to one single company, Charter Communications. From the names of the routers, we can sort of guess where they are. For example, `fyvl` is Fayetteville (the town where I live), `esyr` is East Syracuse, `roch` is Rochester, and `chi` is Chicago. We are traveling west.

```
$ mtr -zb www.ustc.edu.cn
 Host
 1. ---      _gateway (10.0.5.1)
 2. ---      192.168.0.1 (192.168.0.1)
 3. AS11351  142-254-213-109.inf.spectrum.com (142.254.213.109)
 4. AS11351  agg61.fyvlnyhe02h.northeast.rr.com (24.58.240.229)
 5. AS11351  agg78.esyrnyaw02r.northeast.rr.com (24.58.52.86)
 6. AS11351  agg27.rcr01rochnyei.netops.charter.com (24.58.32.76)
 7. AS7843   bu-ether13.tustca4200w-bcr00.tbone.rr.com (66.109.6.2)
 8. AS7843   0.ae0.pr0.chi10.tbone.rr.com (66.109.6.153)
```

At Chicago, we say goodbye to Charter, and hop on `alter.net`, which belongs to Verizon, a national backbone in the US. Through this national "highway", very quickly, we arrive at Los Angeles. That is where we get switched from Verizon to China Unicom, a national backbone in China.

```
 9. AS701    te-0-1-0-1.gw6.chi13.alter.net (152.179.92.69)
10. ... no information is available for this router ...
11. AS701    chinaunicom-gw.customer.alter.net (157.130.230.58)
```

We find ourselves inside a fiber cable laid underneath the Pacific Ocean. Even though we have plenty of light inside the cable, we cannot see anything outside, because our fiber is wrapped with many protection layers, so it is not easy to break. Because of these layers, even if a shark bits the cable, even though that is quite rare, we will still be safe inside. While traveling inside the cable, our light gets weaker and weaker, but every 50 miles, we get a boost, so when we reach the other end of the fiber, our signal is still strong enough. There is also a wire in the layer, and it is used to transmit the electricity to power these boosters.

Finally, we arrive at a landing point, where we get switched into land lines, and the next thing we know, we are in Beijing. Continuing our journey inside China Unicom's networks, eventually, we arrive at Hefei, Anhui, where USTC is located. The entire journey takes 120 milliseconds, with two third of time spent on crossing the Pacific Ocean.

```
12. AS4837   219.158.96.233    Beijing
13. AS4837   219.158.98.93     Beijing
14. AS4837   219.158.8.113     Beijing
15. AS4837   219.158.8.166     Beijing
16. AS4837   219.158.115.34    Beijing
17. AS140726 58.242.195.162    Hefei, Anhui
18. AS140726 58.242.32.130     Hefei, Anhui
19. AS4837   218.104.71.168    Hefei, Anhui (USTC, the destination)
```

The outline of this chapter and the Internet emulator. Throughout the entire journey, we never got lost. At each stop, the router always know where to direct us. How do they know the directions? The BGP protocol plays an essential role in this process. The focus of this chapter is on how BGP works and how it can be attacked. Due to its complexity, we will first discuss the physical infrastructure of the Internet and provide an overview of the BGP protocol. Then we will dive into low-level details of the protocol, showing how exactly BGP works in practice.

To help students gain hands-on experiences on BGP, we spent three years building an Internet emulator, which allows us to run a min-Internet inside a single computer. Because of this emulator, we are not only able to explain the theories behind BGP, but also able to demonstrate how BGP works in action. Integrating the theory and practice is important for understanding BGP, which is very complicated, probably the most complicated topic discussed in this book.

27.2 Physical Infrastructure

In this section, we will look at the physical infrastructure of the Internet, and see the essential physical components of the Internet.

27.2.1 Autonomous Systems

An autonomous system (AS) is a collection of connected Internet Protocol (IP) routing prefixes under the control of one or more network operators on behalf of a single administrative entity or domain [Hawkinson and Bates, 1996]. Each AS is assigned an autonomous system number (ASN). The original ASN numbers only use 16 bits, but that is not enough, so RFC 6793 extended ASN numbers to 32 bits. There are two major types of ASes.

- Stub AS: This type of AS does not provide transit services to others. They are end customers, such as universities, organizations, and most companies. Some stub ASes only connect to one other AS (typically an upstream provider). They are called single-homed stub ASes. Some stub ASes connect to multiple ASes, and they are called multi-homed stub ASes. A multi-home stub AS will not allow traffic from one AS to pass through to another AS, i.e., it does not provide transit service.

- Transit AS: This type of AS connects to multiple ASes, and offer to route data from one AS to another AS. It provides transit services.

27.2.2 Internet Exchange and Peering

An autonomous system needs to connect to other autonomous systems, so they can exchange network traffic. Without the connection, an AS will be an isolated system. Through the connections, users from one autonomous system can reach the users on another AS. Connecting two autonomous systems is called *peering*.

There are two types of peering, public peering and private peering. Public peering usually occurs inside a public facility called Internet Exchange Points (IX). An IX is basically a big high-throughput switch that connects the routers from different ASes (see Figure 27.1(A)). Through this switch, packets from one AS can be handed over to another AS.

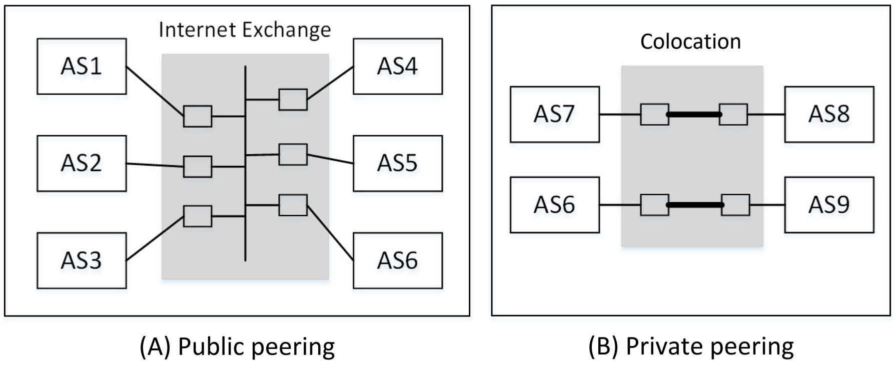

Figure 27.1: Public and private peering

Another type of peering is private peering. When two ASes peer privately, their routers are directly connected via a dedicated cable (fiber or copper), instead of through a switch. This is called *cross connect*. A cross-connection can be viewed as a direct point-to-point connection between two routers. Although private peering can be done inside an IX, most private peering occur at colocation center, which is a special type of data center.

To peer with others at a data center (IX or colocation), an AS needs to put their devices inside the data center. This is called PoP (Point of Presence). A PoP usually includes routers, servers, and switches.

In addition to exchanging network traffics, routers involved in peering also participate in a routing protocol called BGP (Border Gateway Protocol), the goal of which is to exchange destination information, so each autonomous system knows how to route packets to external networks. Understanding how the BGP protocol works and how it can be attacked is the main focus of this chapter.

27.2.3 Laying Cable

An autonomous system's BGP routers at each PoP must connect to its networks, so the inbound traffic received at the PoP location can come into the AS while the outbound traffic can be routed out from the PoP. Typically, PoPs and the networks of an AS are at different physical locations, so cables must be laid to connect them. An autonomous system can lay its own cables, but doing this is quite expensive, especially for small ASes. There are companies who invest in the cable business, and they lay the cable; most individual ASes lease the cable from them, or a channel inside a shared cable.

Sometimes, cables are long distance, and some of them has to be laid at the bottom of an ocean, such as the Pacific Ocean and the Atlantic Ocean. These are called submarine cables. While traditionally, submarine cables were owned by telecom carriers, but recently, more and more content providers are investing in the cable business. For example, Google, Facebook, Microsoft, and Amazon are major investors in new submarine cables. Faced with the prospect of ongoing massive bandwidth growth, owning new submarine cables makes sense for these companies.

For example, Google has many PoPs in Europe to peer with other autonomous systems. This way the traffic to Google can directly get into Google's networks from those PoPs, be routed through Google's dedicated cable underneath the Atlantic ocean, and get into its cloud data centers in the US, instead of going through other transit ASes. This will reduce the delay and increase the bandwidth. It is important for the cloud business. That's why it is not surprising that all the major cloud providers are investing heavily in submarine cables.

27.2.4 Case Studies

In this case studies, we look at three different types of autonomous systems.

A national backbone (a transit AS). To provide transit AS to other ASes, a transit AS need to have PoPs at various locations. Through these PoPs, the transit AS connect with other ASes. A large transit AS has PoPs in many different locations. It lays cables to connect these PoPs to its networks, so traffic entering from one PoP can be routed within its network, eventually reaching another PoP, where the traffic can exit and be given to another AS. Figure 27.2 shows a conceptual example of such a large transit AS.

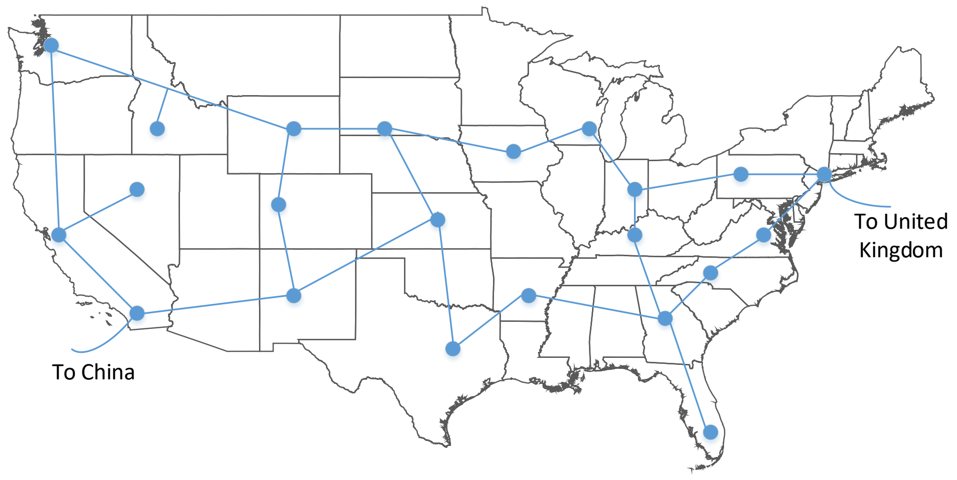

Figure 27.2: A conceptual diagram of a large transit AS

Some of these PoPs are inside big Internet Exchange, and some are in smaller colocation centers. Through these PoPs and the peering, the AS can provide Internet services to its customers, as well as exchanging traffic with other ASes. For example, on the east, the transit AS has a PoP inside the Manhattan Landing Internet Exchange (MAN LAN) in New York City. From there, it peers with many other ASes. This PoP also connects to another PoP at United Kingdom, through a submarine fiber optical cable under the Atlantic Ocean, allowing the transit AS to peer with others in Europe.

On the west, the transit AS connects to a PoP in Hong Kong, China, through submarine fiber optical cable under the Pacific Ocean. It peers with other transit ASes in Asia, pulling the traffic from there to the US, and further deliver them to the corresponding PoPs in the backbone.

A state backbone (a transit AS). NYSERNET is an Internet service provider inside the New York state, and it is mainly for educational purposes, connecting many of the education institutes inside the state. Figure 27.3 shows its PoP map as of 2021.

As we can see, NYSERNET has PoPs in several data centers across the state, so it can peer with the ASes (mostly stub ASes) in those regions. For example, at Syracuse, it peers with the Syracuse University's autonomous system, as well as with many other schools, museums, and research institutes in the regions, providing the transit service to them.

At Buffalo and New York City, NYSERNET peers with other transit ASes, so traffic going out of the New York state can be switched to another AS. One of the peers is the Internet2, which is a national backbone.

A multi-homed stub AS. If organizations, such as a corporation of university, wants to connect to the Internet, they need to purchase services from the Internet Service Providers (ISPs). To provide services, these ISPs typically set up PoPs in a colocation center near the customers, so these organizations can connect to these PoPs.

Let us use Syracuse University (SU) as an example. SU leased fiber cable to connect its BGP routers on campus directly to the circuit in the colocation center at the Syracuse downtown,

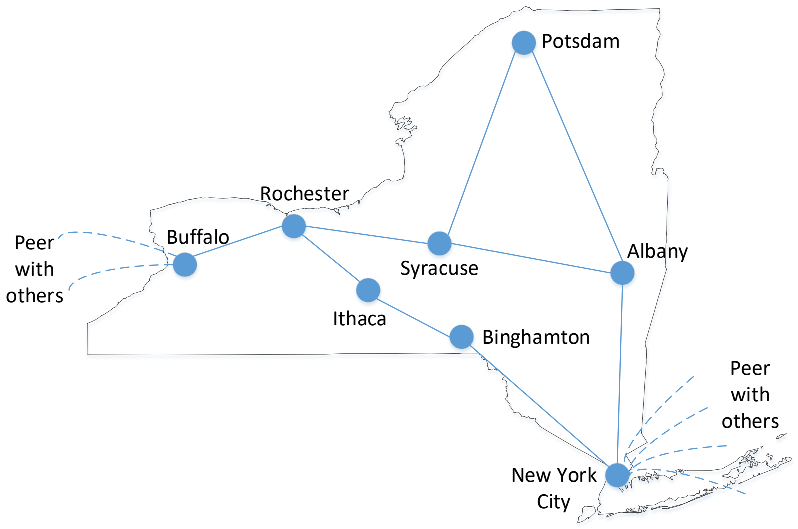

Figure 27.3: NYSERNet

which is 2 miles away from the campus. At there, SU's BGP peers with Charter, Cogent, and NYSERNET, which are the ISPs for SU. These ISPs all have PoPs inside this colocation center.

Many years ago, SU did not peer with RoadRunner, the then Internet service provide for the residents in my neighborhood. My packets to the campus networks had to travel to Philadelphia, where they would get out of RoadRunner and enter Cogent, which routed the traffic to the Syracuse downtown, and handed over the packets to SU's AS. My house is only 10 miles away from the campus. RoadRunner was later purchased by Time Warner, and then by Charter, which does peer with SU at Syracuse. Since then, my packets to the university no longer need to take a huge detour. Therefore, do not be surprised if your packets to your neighbors have to travel hundreds of miles, as you may not be using the same ISP, and the closest data center for these two ISPs to peer is another city.

27.3 The BGP Protocol: Overview

With the cables and peering, the networks from the autonomous systems are interconnected, and the Internet is formed. However, we still have one problem that needs to be resolved: when a packet arrives at an router, which direction should the router forward the packet to? Many routers are connected to more than two networks, so there are multiple directions to route a packet. How do they know which direction should a packet go. The decision will be based on the routing information inside each router, but how do routers get the routing information? That is the purpose of the routing protocols.

27.3.1 Routing Protocols

Let us use highway as an analogy. To provide directions, the transportation department posts signs along the highways. If we consider vehicles as packets, these signs are routing information.

In this case, the routing information is established via manual efforts. Roads are quite static, so manually posting the routing information is practical. However, the Internet is quite dynamic, and routes can change very frequently. That makes the manual effort inadequate.

To get the direction information, routers would constantly talk among themselves, so they can exchange routing information and update one another if routes have changed. This is done through routing protocols. There are many routers on the Internet, so it is impractical for them to participate in the same protocol. There are two levels of routing protocols.

- IGP: (Interior Gateway Protocols): Inside an autonomous system, the internal routers talk among themselves, exchanging routing information, so routers know where to route packets within the AS. These routers only talk to the routers within the same AS, and they do not talk to the outsider. Examples of IGP includes OSPF, RIP, and IGRP. How IGP works is beyond the scope of this chapter.

- EGP (Exterior Gateway Protocols): It is used for determining network reachability between autonomous systems. When an packet exits an autonomous system, the router at the exit point (or edge) needs to know which AS the packet should be given to. The edge routers from different ASes participate in this protocol to exchange the network reachability information. There is only one widely-adopted EGP protocol, and that is BGP, Border Gateway Protocol. This chapter focuses on BGP.

27.3.2 The BGP Protocol: A High-Level Explanation

The actual BGP protocol is quite complicated, so we break it down into several sections. In this section, we only give a high-level explanation of how the protocol works. In later sections, we dive into the details of each aspect of BGP.

BGP speakers and routers. Each AS will designate one or multiple devices as its representatives, called BGP speakers. They peer with other BGP speakers, so they can exchange routing information with the peers. Through this protocol, a speaker will know the path to a particular destination. After getting the external routing information, the speaker disseminate the information to the routers inside the AS, so the internal routers also know which path they should take when routing to an external destination.

The role of BGP speakers is to gather routing information and provide the information to routers, they do not necessarily route packets, so BGP speakers and BGP routers are different entities. However, in most situations, they are both on the same devices. In this case, BGP routers are often used, instead of BGP speakers. We primarily use BGP routers in this chapter.

BGP peering. For two BGP routers to exchange routing information, they need to peer with each other. Figure 27.4 shows how the BGP routers A - G peer with one another. A BGP router can peer with a BGP router from a different AS or from the same AS. If they are from different ASes, the protocol running between them is called EBGP (External BGP). If they are from the same AS (e.g. D and E), the protocol is called IBGP (Internal BGP). Using the peering and the BGP protocol, these BGP routers exchange routing information.

IP prefix announcement. Each AS owns a set of network IP prefixes, and it has to announce them to the rest of the Internet, telling others that to reach these IP prefixes, come to this particular AS. Let us use AS150 as an example. Assuming it owns a network with prefix

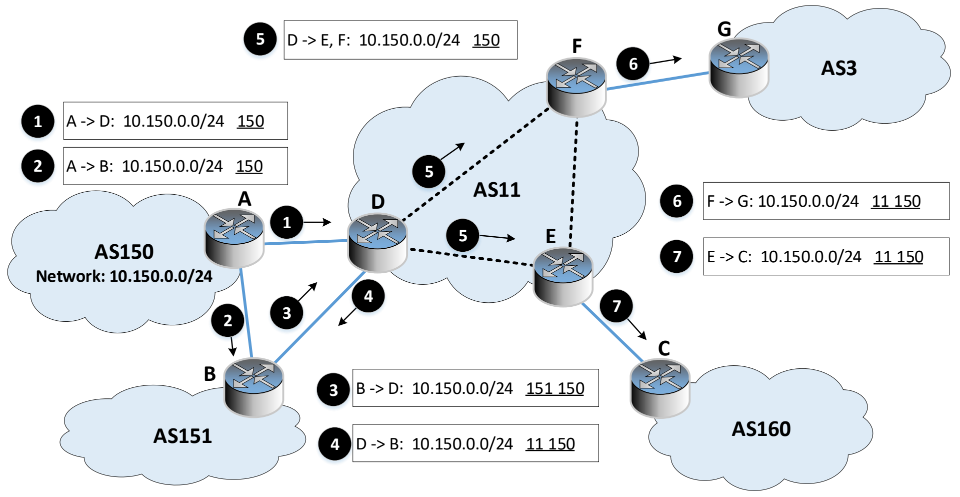
5 D -> E, F: 10.150.0.0/24 150
1 A -> D: 10.150.0.0/24 150
2 A -> B: 10.150.0.0/24 150
AS150
Network: 10.150.0.0/24
A
B
D
E
F
G
C
AS11
AS3
AS151
AS160
6 F -> G: 10.150.0.0/24 11 150
7 E -> C: 10.150.0.0/24 11 150
3 B -> D: 10.150.0.0/24 151 150
4 D -> B: 10.150.0.0/24 11 150

Figure 27.4: How BGP works

`10.150.0.0/24` (see Figure 27.4). It has to announce this prefix to its peers (BGP routers B and D); otherwise, nobody knows about it. In the announcement, for each destination, a path is provided. This path, called *AS path*, lists all the ASes that need to be traversed to reach the destination. In the example, the AS path for `10.150.0.0/24` announced by AS150 only contains `150`, the ASN of the origin. In the figure, ❶ and ❷ are what AS150 announces to D and B, respectively.

Route selection and forwarding. When Routers B and D receive the prefix announcement from AS150, they will further announce this prefix to their peers. Let us first look at what BGP Router B in AS151 will do to this prefix. After receiving A's announcement, B now knows a path to the network `10.150.0.0/24`. It further announces this path to its own peers (other than A, because the path is learned from A). In the announcement, B adds its own ASN to the front of the path, so the AS path sent from B to D is `"151 150"` (see ❸ in the figure).

Now, let us look at what Router D in AS11 will do to the prefix `10.150.0.0/24`. D receives two AS paths for this prefix, i.e., there are two different paths to reach this network.

- From Router A in AS-150 (❶): the AS path is `"150"`
- From Router B in AS-151 (❸): the AS path is `"151 150"`

D will store both AS paths, but will select only one to announce to its peers as the best route to `10.150.0.0/24`. The path selection follows an algorithm, which will be discussed in details in § 27.8. The length of the AS path is one of the selection criteria. Assuming that in this case, the length criterion is used, so the shorter AS path `"150"` is selected. Router D will then forward this route to its peers. Here is what D will do to all its peers:

- Peer B is an EBGP peer, so D will announce the destination `10.150.0.0/24` to it with an updated AS path `"11 150"`, i.e., adding its own ASN to the front of the AS path (❹ in the figure).

- Peers E and F are IBGP peer, so D will announce the destination to them without adding its own ASN to the AS path (❺ in the figure). Although E and F do not peer with any BGP router from AS150, through the IBGP peering with D, they now know a path to `10.150.0.0/24`.
- Peer A is from AS150, which is already on the AS path, so D will not forward the selected path to A. This is to avoid loop.

Routers E and F will further announce the network prefix to their EBGP peers G and C, respectively. In their announcement, the ASN `11` will be added to the AS path (see ❻ and ❼ in the figure). It should be noted that E will not forward the network prefix to its IBGP peer F, and vice versa. BGP routers do not forward the route to their IBGP peers if the route is learned from an IBGP peer (the explanation on this is provided in § 27.10.1).

We have only explained how the IP prefix `10.150.0.0/24` in AS150 is propagated on the Internet. Network prefixes belonging to other ASes are propagated in a similar fashion. The explanation here is at a high level, and many details of the process are not shown. In the subsequent sections, we will dive into those details.

Path withdrawal. If for some reason, the link between A and D is broken, D can no longer reach `10.150.0.0/24` from that link, i.e. through the AS path `"150"`, so it will send an update message to all its peers to withdraw that route. Moreover, D will re-run its path selection algorithm. This time, the best AS path to reach `10.150.0.0/24` is `"151 150"`, which is the one received from Router B. Therefore, D will announce this new route to its peers.

If the link between A and D is not broken, but the link between B and D is broken, that will not affect the route selected by D with regarding to the destination network `10.150.0.0/24`, so D will not send any update message regarding this network.

27.4 The SEED Internet Emulator

Due to the complexity of BGP, it is hard to learn BGP simply from readings. It is important for readers to get a first-hand experience on BGP. I have taught BGP for many years, but could not find any platform that I could use for students to gain hands-on experience on BGP. This is mainly because BGP involves many computers and networks in a realistic environment. Building such an environment is not easy. That is the main reason why we spent 3 years developing the SEED Internet Emulator (simply called the Emulator in this chapter). A series of video tutorials on the Emulator can be found from YouTube (click here).

27.4.1 The SEED Internet Emulator

The demonstration and experiments described in this chapter are based on a pre-built Internet emulator, which is provided to readers inside the `Labsetup.zip` file. The files inside the `output` sub-folder are the actual emulation files (container files), and they are generated from the Python code `mini-internet.py`, which is also included. However, to run this program, one needs to install the Emulator source code from GitHub (click here). The advantage of the Python code is that if readers want to modify the Emulator, such as adding a new autonomous system, a new peering, a new network, etc., they can easily do that in the Python code. Detailed manuals for the Emulator can be found from the project's GiHub repository.

Start the emulation. Go to the `Labsetup/output` folder, run the following docker commands to build and start the containers. It is better to run the Emulator inside the provided SEED Ubuntu 20.04 VM, because everything that the Emulator depends on has already been installed. However, users can also use a generic Ubuntu 20.04 build as long as all the needed software packages are installed.

```
$ docker-compose build
$ docker-compose up
```

The network map. Each computer (hosts or routers) running inside the Emulator is a docker container. Users can access these computers using docker commands, such as getting a shell inside a container. The Emulator also comes with a web application, which visualizes all the hosts, routers, and networks. After the Emulator starts, the map can be accessed from this URL: `http://localhost:8080/map.html`. Users can interact with this map, such as getting a terminal from a container, disabling BGP sessions, and setting filters to visualize network traffic. The syntax of the filter is the same as that in `tcpdump`, because the filter is actually given to the `tcpdump` program running inside each container.

Running `tcpdump`. During the experiments, we may need to capture and display packets. While `tcpdump` can do these, for displaying packets, it is better to use Wireshark. Unfortunately, it is very hard to run GUI-based applications, such as Wireshark, inside a container. Therefore, we use a hybrid approach. Inside the container, we use `tcpdump` to capture packets, and save them into a `pcap` file. We run the following command inside the container to capture BGP packets (port `179` is used for BGP). The captured packets will be stored in the file `/tmp/bgp.pcap`.

```
# tcpdump -i any -w /tmp/bgp.pcap "tcp port 179"
```

We then copy the file to the host machine, and open it using Wireshark. Copying files between a container and the host machine can be done using the "`docker cp`" command on the host machine.

```
$ docker cp <container id>:/tmp/bgp.pcap ./bgp.pcap
```

27.4.2 BIRD: The Routing Software Used in the Emulator

The routing software used in the Emulator is called BIRD, an acronym standing for "BIRD Internet Routing Daemon" [CZ.NIC, 2021]. BIRD is open source under the GNU General Public License. It supports a number of standard routing protocols, including the Border Gateway Protocol (BGP), the Routing Information Protocol (RIP), and the Open Shortest Path First protocol (OSPF). We mainly use BGP and OSPF in the Emulator. We will configure these routing protocols in BIRD, so it is important to understand how BIRD configures its routing protocols.

BIRD takes the configuration file `bird.conf` from the `/etc/bird/` folder. The actual configuration for BIRD is quite complicated, and most configuration entries are protocol specific. Full details of the configuration can be found in the BIRD manual [CZ.NIC, 2021]. We only focus on the features used by the Emulator.

BIRD's routing tables. The purpose of most routing protocols is for protocol participants to exchange information with their peers. The information will be stored in tables, which are called routing tables. BIRD has several routing tables in memory. How these routing tables receive or export their data are specified as a `protocol`. In some cases, these are real routing protocols, such as BGP, OSPF, and RIP. Namely, the routing table connects to a router, and the routes received from the router via the specified protocol are imported to the routing table, while the routes stored in the routing table are exported to the router.

Some protocols used in BIRD are not real protocols. BIRD uses the keyword `protocol` in a broader sense, specifying where a BIRD routing table gets its routes from, and where the routes in the table need to be sent to.

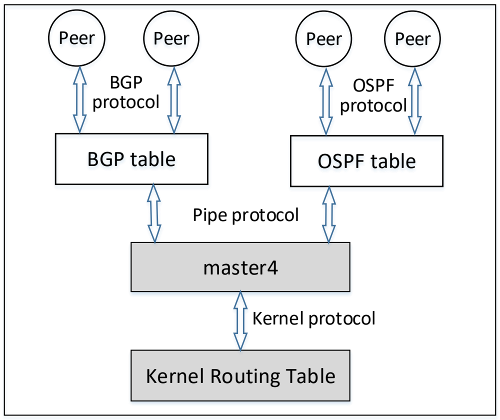

Figure 27.5: Relationships among the tables in BIRD

These special "protocols" are used to specify how data flow from one routing table to another. Inside the Emulator, on each BGP router, we save the routes from different protocols in separate tables. For example, routes obtained from the BGP protocol are stored in the table `t_bgp`, and the routes from the OSPF protocol are stored in the table `t_ospf`. Using the `pipe` "protocol", we can feed the routes from these tables to BIRD's master table (`master4` for IPv4). The routes in the master table will eventually get into the kernel routing table, which is the actual routing table used by the operating system for packet forwarding. Figure 27.5 depicts the relationships of these tables. We will explain them in further details in the rest of this section.

27.4.3 Pipe Between Tables

For IPv4, only the route stored in BIRD's master table `master4` will be eventually exported to the kernel's routing table (via the `kernel` protocol). If we do not specify a table name in a protocol, the default one used will be the master table. However, in our configuration, to make it easier to manage routes, we do specify a customized table for different protocols. We need to get these routes into the master table. This is done through a pipe, which is another special type of protocol in BIRD.

The `pipe` protocol links two routing tables, a primary table (specified using the `table` keyword) and a secondary table (specified using the `peer` keyword). Filters can be specified for the import and export directions. In the following example, all the entries in the `t_bgp` table are exported to the `master4` table, while none of the routes in the `master4` are imported.

```
protocol pipe {
    table t_bgp;
    peer table master4;
    import none;
    export all;
}
```

In the following example, all the route entries from the `t_direct` table are exported to the `t_bgp` table, and the local preference attribute of the routes are set to `40`.

```
protocol pipe {
    table t_direct;
    peer table t_bgp;
    import none;
    export filter { bgp_local_pref = 40; accept; };
}
```

27.4.4 BGP Routing Table vs. Kernel Routing Table

None of the BIRD routing tables is used for the actual routing. The one actually used for routing is called kernel routing table, which is inside the operating system kernel. However, the kernel routing table depends on the routing protocols to learn the actual routes.

In BIRD, a special protocol called `kernel` protocol is used to connect BIRD's routing tables to kernel's routing table (see Figure 27.5). An example is given in the following. In this example, no table is specified, so the table used is the master table (`master4` for IPv4). In the example, `"import all"` means BIRD will import everything from the kernel's routing table to its `master4` table; `"export all"` means all the routes in the `master4` table will be exported to the kernel's routing table. This is how BGP routers set the routing table using the data collected from the BGP protocol and other routing protocols.

```
protocol kernel {
    ipv4 {
        import all;
        export all;
    };
    learn;
}
```

Experiment. Let's do an experiment by modifying BIRD's configuration file `/etc/bird/bird.conf` on a BGP router inside the Emulator. We change `"export all"` to `"export none"` in the `kernel` protocol. This means none of the routes from BIRD will be exported to the kernel routing table. After making the changes, reload the configuration (the first command), and then check the kernel routing table (the second command).

```
# birdc configure
BIRD 2.0.7 ready.
Reading configuration from /etc/bird/bird.conf
Reconfigured
```

```
# ip route
10.100.0.0/24 dev ix100 proto kernel scope link src 10.100.0.150
10.150.0.0/24 dev net0 proto kernel scope link src 10.150.0.254
```

Before the change, the BGP router knows how to reach the other networks in the Emulator, and the the output of `"ip route"` has many entries. After the change, we can clearly see that the kernel now knows nothing about the rest of the Emulator. This is because the kernel does not get any route from BIRD.

27.4.5 The Mandatory `device` Protocol

Each BGP router has a mandatory protocol called `device`. This is not a real routing protocol. It does generate any route, nor does it accept any route. It is only used to get information about the network interfaces from the kernel. Without the `device protocol`, BIRD knows nothing about the network interfaces, so it will not be able to run BGP, as it does not know how to reach peers. Therefore, this protocol is mandatory. It is just an empty block in our Emulator.

```
protocol device {   }
```

27.5 BGP: IP Prefixes Owned By AS

In the BGP protocol, each BGP router needs to tell the Internet the network prefixes (or IP prefixes) owned by the autonomous system that it is representing, telling others that this AS is the origin (owner) of these IP prefixes, and packets to these prefixes should be routed into this AS. How does a BGP router know what IP prefixes are owned by its AS? There are several ways for a BGP router to get that information:

- From the networks that the BGP router is attached to.
- From the entries statically added to the configuration file.
- From the IBGP (Interior BGP) and IGP (Internal Gateway Protocol), i.e., from other routers inside the same AS. These two protocols will be covered in § 27.10.

27.5.1 Route Generation Using the `direct` Protocol

Each BGP router is attached to one or multiple internal networks within the AS, so it knows the IP prefixes for these networks from its own network interfaces. BIRD uses a `direct` protocol to collect such IP prefix information, and generate the corresponding routes to these prefixes. This is not a real protocol, but a route generator for all the directly connected networks. The following is an example from the BIRD configuration file:

```
protocol direct local_nets { # Give it a customized name: local_nets
  ipv4 {
    table t_direct;

    # Import all the generated routes to t_direct.
    import all;
  };
  # Generate routes from these two interfaces
  interface "eth0";
  interface "eth1";
}
```

In the `direct` protocol, the *interface* keyword is used to generate routes from an interface. Assuming that a BGP router uses `eth0` and `eth1` to connect to the autonomous system's internal networks `10.150.0.0/24` and `10.150.1.0/24`, respectively, the `direct` protocol block in the example above will generate routing entries for `10.150.0.0/24` and `10.150.1.0/24`. These routes will be stored in the `t_direct` table. Later we will see that through a pipe, the routes in this table are exported to the BGP table `t_bgp`, and are thus used by the BGP protocol.

Experiments on the `direct` protocol. In our emulation, AS-150 has one network, so there is only one interface listed in the `direct` protocol. We can run the following command on the BGP router to see the status of this protocol.

```
# birdc show protocols all local_nets
BIRD 2.0.7 ready.
Name       Proto      Table      State  Since         Info
local_nets Direct     ---        up     13:51:05.188
  Channel ipv4
    State:          UP
    Table:          t_direct
    Preference:     240
    Input filter:   ACCEPT
    Output filter:  REJECT
    Routes:         1 imported, 0 exported, 1 preferred
    ...
```

As we will show later, the routes stored in the `t_direct` table will eventually be exported to the BGP table `t_bgp`, and then be further exported to the BGP's mater routing table `master4`. We can use the "`birdc show route`" command to list the entries in each table (if no table name is specified in the command, the `master4` table will be used by default).

```
# birdc show route all table t_direct 10.150.0.0/24
Table t_direct:
10.150.0.0/24        unicast [local_nets 13:51:05.189] * (240)
   dev net0
   Type: device univ

# birdc show route all table t_bgp 10.150.0.0/24
Table t_bgp:
10.150.0.0/24        unicast [local_nets 13:51:05.189] * (240)
   dev net0
   Type: device univ
   BGP.local_pref: 40
   BGP.large_community: (150, 0, 0)

# birdc show route all table master4 10.150.0.0/24
Table master4:
10.150.0.0/24        unicast [local_nets 13:51:05.189] * (240)
```

```
   dev net0
   Type: device univ
   BGP.local_pref: 40
   BGP.large_community: (150, 0, 0)
```

27.5.2 Routes Generated From the `static` Protocol

We can also add static IP prefix information in the BGP configuration file. In BIRD, this is done via the `static` protocol. This is not a real protocol either, as it does not import routes from a peer; instead, it provides predefined routes that need to be imported into the routing table.

When specifying a route, we need to specify a target to indicate the action that needs to be performed. The target can be a router, so the packets to the destination should be forwarded to this router. There is an interesting target name called `blackhole`, indicating that the packets to the destination should be dropped. In addition, each route can have a route-specific filter. This is especially useful for configuring route attributes. Filters will be discussed in § 27.6.2.

Let us add the following `static` protocol to AS-150's BGP router. After making changes to the configuration file `/etc/bird/bird.conf`, we need to run `"birdc configure"` to reload the configuration. Once the routes are reloaded, the BGP router will announce them to the outside. If we go to other BGP routers inside the emulator, check their routing tables using `"ip route"`, we can clearly see that the `10.150.1.0/24` and `10.150.2.0/24` networks are in their routing tables.

```
protocol static {
    ipv4 {
        table t_bgp;
    };
    route 10.150.1.0/24 via 10.150.0.254 {
             bgp_large_community.add(LOCAL_COMM);
    };
    route 10.150.2.0/24 blackhole {
             bgp_large_community.add(LOCAL_COMM);
    };
}
```

27.5.3 ASN and Its IP Prefixes

In the real world, sometimes we would like to know what IP prefixes are owned by a particular AS, and what AS is the owner for a particular IP prefix. These can be done using the RADB database. The following command lists all the IP prefixes owned by AS11872, the ASN belonging to the Syracuse University.

```
$ whois -h whois.radb.net -- '-i origin AS11872' | grep route:
route:       128.230.0.0/16
route:       149.119.0.0/16
route:       128.230.0.0/17
route:       128.230.128.0/17
route:       128.230.0.0/18
route:       128.230.64.0/18
route:       128.230.0.0/19
```

```
route:       128.230.32.0/19
route:       149.119.0.0/17
route:       149.119.128.0/17
route:       149.119.0.0/18
route:       149.119.64.0/18
```

We can also find the owner of a particular IP prefix. The following command shows the ownership of the IP address `31.13.78.3`.

```
$ whois -h whois.radb.net 31.13.78.3
route:       31.13.78.0/24
descr:       Facebook, Inc.
origin:      AS32934
mnt-by:      MAINT-AS32934
changed:     shaw@fb.com 20120423  #20:09:37Z
source:      RADB
```

27.6 BGP Peering

In this section, we show exactly how BGP routers peer with each other. We use the Emulator to demonstrate how the peering works. In the Emulator, peering is conducted in Internet eXchanges (IX), each of which is emulated using a local network. An AS that peers with other ASes at an IX needs to have a PoP inside the IX. This is emulated as having a BGP router connected to the network provided by the IX.

Figure 27.6 depicts the setup at IX-100, where AS-150, AS-151, AS-2, AS-3, and AS-4 are connected to the IX-100's network `10.100.0.0/24`. These autonomous systems all have a BGP router connected to the network, so they are physically connected. That does not mean they peer with one another. If two BGP routers want to peer with each other, they need to set up a peering relationship.

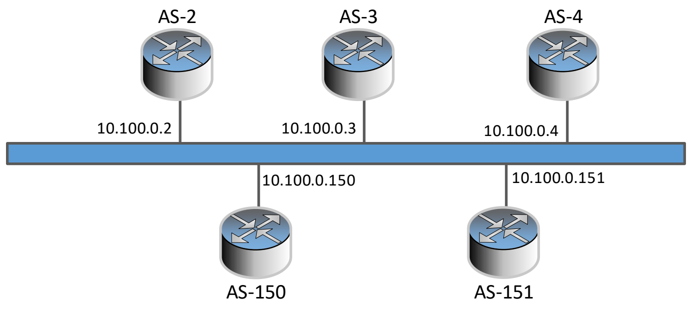

Figure 27.6: The setup of the Internet exchange IX-100

27.6.1 Establishing Peering Relationship

After two BGP routers are physically connected, either through a public peer or a private peering, they need to set up a BGP peering between themselves, so they can talk to each other using

BGP and exchange route information. A BGP router can peer with multiple peers, and each one is called a *peering session*. In BIRD, each peering session is defined using a protocol called `bgp`. In the Emulator, AS-150 peers with AS-2 at IX-100. The following example shows the peering configuration on AS-150's BGP router.

```
protocol bgp u_as2 {
    ipv4 {
        table t_bgp;
        import filter {
            ... omitted ...
        };
        export where ... filter omitted ...
        next hop self;
    };
    local    10.100.0.150 as 150;
    neighbor 10.100.0.2   as 2;
}
```

The `local` option. It specifies which AS the router belongs to and the IP address of the router. This IP address should be the one on the IX's network (IX-100's network is `10.100.0.0/24` in the Emulator). The IP address part is optional, but it makes the configuration looks clearer and can prevent selecting the wrong IP address for the BGP session when there are multiple IP addresses on the router.

The `neighbor` option. It specifies the IP address of the peer and what AS it belongs to. This is the actual peering part. In this example, we set up a BGP session between `AS-150` and `AS-2`, so they can exchange route information using the BGP protocol. Similarly, the IP address here should be the one on the IX's network.

The IPv4 channel and the routing table. Each protocol is connected to a routing table through a channel. BGP supports both IPv4 and IPv6 channels. Each channel has two filters, export and import filters, which can accept, reject and modify the routes. The export filter applies to the routes going from the routing table to the protocol, while the import filter applies to the routes coming into the routing table. In the example above, the routing table in the IPv4 channel is specified (`t_bgp`). If no table is specified, the default table used is `master4` (for IPv6, it is `master6`).

27.6.2 Import and Export Filters

When importing/exporting routes to/from the routing table, filter rules can be applied. BIRD contains a simple programming language, so filter rules are programs. The syntax of the programming language can be found from the BIRD manual [CZ.NIC, 2021]. When a route is being passed between protocols and routing tables, the corresponding filter will be interpreted by BIRD. The filter will get the route, so it can inspect the route, the attributes, and make changes to the route if needed. At the end, the filter decides whether to pass the route through (using `accept`) or reject it (using `reject`).

The following example shows the filters used in the peering session between AS-150 and AS-2 (omitted in the earlier example). In this example, when routes are imported from the peer (i.e., from AS-2's BGP router), a community information is added and the local preference value is set to `10`. When routes are exported to the peer, only the routes belonging to the `LOCAL_COMM` and `CUSTOMER_COMM` communities can be exported. We will discuss the communities later.

```
table t_bgp;
import filter {
    bgp_large_community.add(PROVIDER_COMM);
    bgp_local_pref = 10;
    accept;
};
export where bgp_large_community ~ [LOCAL_COMM, CUSTOMER_COMM];
```

A filter is passed a route, so it can access the attributes of the route via predefined variables. We list some of the useful variables in the following, while the detailed list can be found from the BIRD manual [CZ.NIC, 2021].

- `net`: the network prefix of the route.
- `bgp_path`: the AS path of the route.
- `bgp_local_pref`: the local preference value of the route; used for path selection.
- `bgp_next_hop`: the next hop used for forwarding packets to this destination.
- `bgp_large_community`: list of large community values associated with the route.

27.6.3 Peering via Route Server

In a public Internet exchange, autonomous systems want to peer with many other autonomous systems. Let's say we have N autonomous systems, and they want to peer with one another. If we use the approach described earlier, each pair of ASes needs to set up a peering relationship. That will be quite complicated.

Most Internet exchanges provide a mechanism to simplify this. They provide a special server called *route server*. All these N autonomous systems will only need to peer with this route server. When the router server receives a route from a participant over BGP, it re-distributes the routes to all other connected participants. The route server function pretty much like multicast: any BGP route sent to the router server will be received by everybody that peers with the route server.

It should be noted that route server is not a real BGP peer, and its behavior is different from a real BGP peer. Most importantly, it does not add its own ASN to the path, nor does it change the `next-hop` attribute of the route. It is transparent to other BGP routers, and will not affect the outcome of the BGP protocol. Peering via a route server is equivalent to peering directly. Its main purpose is solely to make peering among many BGP routers easier.

In our Emulator, AS-2, AS-3, and AS-4 peer with one another at IX-100. Instead of peering directly, they use the route server approach. If we go to AS-2's BGP router at IX-100, we can see the following BGP entry in the configuration file. It creates a peer with `10.100.0.100`, which is the route server provided by IX-100. AS-3's and AS-4's BGP routers do the same.

```
protocol bgp p_rs100 {
  ipv4 {
    table t_bgp;
    import filter {
        bgp_large_community.add(PEER_COMM);
        bgp_local_pref = 20;
        accept;
    };
    export where bgp_large_community ~ [LOCAL_COMM, CUSTOMER_COMM];
    next hop self;
  };
  local    10.100.0.2   as 2;
  neighbor 10.100.0.100 as 100;    ← Peer with the route server
}
```

On the router server, `10.100.0.100`, a peering entry is needed for each of AS-2, AS-3, and AS-4, with the "`rs client`" option turned on. All these "`protocol bgp`" entries will have the same content, except for the `neighbor` option. Here is the example for the peering with AS-2.

```
protocol bgp p_as2 {
    ipv4 {
        import all;
        export all;
    };
    rs client;
    local    10.100.0.100 as 100;
    neighbor 10.100.0.2   as 2;    ← Peer with AS-2
}
```

27.7 BGP UPDATE Message

When two BGP routers peer with each other, they use a TCP connection to establish a BGP session between themselves. The default port number for the BGP protocol is `179`. The peers then send BGP messages to each other using the connection. BGP messages use a fixed-size header, which includes a type field, indicating what type of message it is. There are five types of BGP messages:

- Open message: for establishing BGP connections
- Update message: for transferring routing information between BGP peers.
- Keepalive message: for checking whether the peers are still reachable.
- Notification message: for notifying BGP peers of errors.
- Route-refresh message: a message type to support the Route Refresh Capability.

The BGP UPDATE message is the most important message, so we will only focus on this type in this chapter. The format of the BGP UPDATE message is depicted in Figure 27.7. The number in the parentheses is the number of bytes for that field. Fields without a length specification has a variable length. As we can see, an UPDATE message contains two types of update information: route withdrawals (for withdrawing previously announced routes) and route advertisements (for new or updated routes).

27.7.1 Route Withdrawal

When a BGP's link to a peer is broken, the peer is no longer reachable directly, so all the routes coming from that link are no longer valid. If any of these routes happens to be selected as

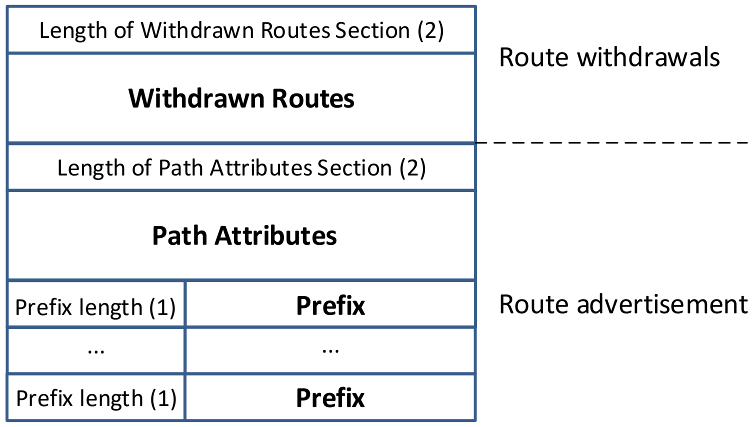

Figure 27.7: BGP Update message

the best route advertised to the other peers, the BGP router needs to withdraw that route. The withdrawal may trigger a chain effect, starting from the peers, to their peers, and so on.

Experiment. Let us intentionally disable the BGP session between AS-164 and AS-12 from the map (there is a button for this). Before doing so, we start the `tcpdump` program on AS-150's BGP router to capture BGP packets.

```
// Display the captured packets directly
# tcpdump -nti any -vvv "tcp port 179"

// Save the captured packets to a PCAP file
# tcpdump -i any -w /tmp/bgp.pcap "tcp port 179"
```

We can either display the captured packets using `tcpdump` inside the container (the first command), or save the captured packets to a `pcap` file and then use Wireshark outside of the container to display the packets (see § 27.4.1). For reader's convenience, the file (`bgp.pcap`) used in this experiment is also included in the emulation folder.

The breaking of the link between AS-164 and AS-12 has triggered route withdrawals throughout the Emulator. This is because AS-12 is the only service provider for AS-164, so if that peering link is broken, nobody knows how to reach AS-164. Therefore, all the autonomous systems will withdraw that route. The following is the route withdrawal message sent from AS-3 to AS-150 (Packet 9 in `bgp.pcap`).

```
Internet Protocol Version 4, Src: 10.100.0.3, Dst: 10.100.0.150
Transmission Control Protocol, Src Port: 179, Dst Port: 33905, ...
Border Gateway Protocol - UPDATE Message
    Marker: ffffffffffffffffffffffffffffffff
    Length: 27
    Type: UPDATE Message (2)
    Withdrawn Routes Length: 4
    Withdrawn Routes
        10.164.0.0/24
```

```
            Withdrawn route prefix length: 24
            Withdrawn prefix: 10.164.0.0
    Total Path Attribute Length: 0
```

This BGP UPDATE message only contains a route withdrawal message. The length of the path attribute field is zero, indicating that there is no route advertisement in this update.

Facebook outage. On October 4th, 2021, Facebook, including Instagram and WhatsApp, suffered a massive outage, which took nearly six hours to resolve. The cause of the outage involved BGP route withdrawals. According to Janardhan [2021], "During one of these routine maintenance jobs, a command was issued with the intention to assess the availability of global backbone capacity, which unintentionally took down all the connections in our backbone network, effectively disconnecting Facebook data centers globally." Although this by itself was a big problem, it should not take very long to fix. It was the secondary issue that made the situation much worse.

When Facebook's DNS servers found out that they could not speak to the data centers, according to the design, their corresponding BGP speakers withdrew the routes to these DNS servers. That essentially took Facebook's DNS servers off the Internet. They were still operational, but nobody knew how to reach them. DNS is so essential to the Internet that without it many of the internal tools used by Facebook could not work, making it very hard for the investigation and communication to be conducted by Facebook engineers.

27.7.2 Route Advertisements

After a BGP router establishes a session with each of its peers, it advertises all its selected routes to the peers. After that, it only advertise incremental updates. For example, if the BGP router has learned a better route to a destination, it sends an update to its peers, advertising this new route. Whether a peer chooses this route or not is up to the peer. If the peer does not choose this route, then the peer will not further advertise the route. However, if the peer indeed chooses this route as its best route, it will further advertise the route to its own peers. Each route advertisement contains two sections:

- Path attributes: This is a list of attributes associated with the path. Each networking equipment vendor can create their own BGP attributes, but some attributes are mandatory, including Origin, AS path, and Next hop.
  - Origin: defines the origin of routing information, i.e., how a route became a BGP route. For example, is the route learned from an internal or external routing protocol?
  - AS path: the list of autonomous systems that need to be traversed to reach the specified destination.
  - Next hop: the next-hop router for the path. If a peer picks this route, it should route packets to this next-hop router.
- Network Layer Reachability Information (NLRI): This is the IP prefix that can be reached from the specified path. Multiple networks may have the same path and path attributes, so the NLRI section can contain multiple network prefixes.

Experiment 1. Let us take a look at a concrete route advertisement message. Previously, we have disabled the BGP session between AS-164 and AS-12. Now let us enable it and observe the BGP traffic triggered by the action. We observe from AS-150's BGP router using `tcpdump`.

The enabling of the BGP session triggers a series route advertisements across the Internet. First, AS-164's BGP router will advertise to its peer AS-12 the prefix `10.164.0.0/24`, which is owned by AS-164 in the Emulator. The peer will then further advertise the route to its own peers, adding its ASN to the front of the AS path, and so on. Eventually, when the route advertisement reaches AS-150, we see the following BGP UPDATE message (Packet 17):

```
Internet Protocol Version 4, Src: 10.100.0.3, Dst: 10.100.0.150
Transmission Control Protocol, Src Port: 179, Dst Port: 33905, ...
Border Gateway Protocol - UPDATE Message
    Marker: ffffffffffffffffffffffffffffffff
    Length: 94
    Type: UPDATE Message (2)
    Withdrawn Routes Length: 0
    Total Path Attribute Length: 67
    Path attributes
        Path Attribute - ORIGIN: IGP
        Path Attribute - AS_PATH: 3 12 164
        Path Attribute - NEXT_HOP: 10.100.0.3
        Path Attribute - LARGE_COMMUNITY: 3:1:0 12:1:0 164:0:0
    Network Layer Reachability Information (NLRI)
        10.164.0.0/24
            NLRI prefix length: 24
            NLRI prefix:  10.164.0.0
```

From this UPDATE message, we can see a path to AS-164's network `10.164.0.0/24`. It goes through AS-3 and AS-12, before reaching AS-164. If this route is selected by AS-150 for routing, packets going from AS-150 to the destination will be routed to the next-hop router `10.100.0.3`, which is a BGP router in AS-3.

Experiment 2. Let us stop AS-150's BGP router and then start it again (the commands are listed below). From the captured BGP packets, we can see the OPEN message type (e.g., Packets 66 and 68). That is when the BGP router is establishing BGP sessions with the peers. After the sessions are established, the peers will advertise the routes that they know to AS-150. For example, in Packet 78, we can see a list of UPDATE messages were sent from AS-2 to AS-150; in Packet 101, a list of UPDATE messages were sent from AS-3 to AS-150.

```
// Commands to shutdown and restart the BIRD daemon (on AS-150)
# birdc down
# service bird start

// Route advertisement from AS-2 to AS-150
Internet Protocol Version 4, Src: 10.100.0.2, Dst: 10.100.0.150
Transmission Control Protocol, Src Port: 53365, Dst Port: 179, ...
Border Gateway Protocol - UPDATE Message
Border Gateway Protocol - UPDATE Message
... (omitted) ...
Border Gateway Protocol - UPDATE Message
Border Gateway Protocol - UPDATE Message  ← This one is expanded
```

```
    Marker: ffffffffffffffffffffffffffffffff
    Length: 82
    Type: UPDATE Message (2)
    Withdrawn Routes Length: 0
    Total Path Attribute Length: 51
    Path attributes
        Path Attribute - ORIGIN: IGP
        Path Attribute - AS_PATH: 2 4
        Path Attribute - NEXT_HOP: 10.100.0.2
        Path Attribute - LARGE_COMMUNITY: 2:2:0 4:0:0
    Network Layer Reachability Information (NLRI)
        10.4.0.0/24
        10.4.1.0/24
Border Gateway Protocol - UPDATE Message
Border Gateway Protocol - UPDATE Message
```

27.7.3 TTL and BGP TTL Security Hack

One thing that we might notice is that in the IP header of all the BGP UPDATE messages, the Time-To-Live (TTL) field is always 1, indicating that the life span of the packet is only one hop. Therefore, the packet is only visible within the sender's direct network, and it will not be able to routed out. Why?

```
Internet Protocol Version 4, Src: 10.100.0.2, Dst: 10.100.0.150
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0xc0 (DSCP: CS6, ECN: Not-ECT)
    Total Length: 146
    Identification: 0x5a0f (23055)
    Flags: 0x4000, Don't fragment
    Fragment offset: 0
    Time to live:  1
    Protocol: TCP (6)
    Header checksum: 0x0938
    Source: 10.100.0.2
    Destination: 10.100.0.150
Transmission Control Protocol, Src Port: 53365, Dst Port: 179, ...
Border Gateway Protocol - UPDATE Message
```

EBGP peering typically requires two BGP routers to connect directly, such as through a LAN or a switch. Setting TTL to 1 ensures that only the directly connected peers can receive the UPDATE messages.

BGP TTL security hack. Setting TTL to 1 prevents the UPDATE message from going out, but in terms of security, it is more important to prevent forged UPDATE messages from getting into the peering network. Since peering is for locally connected BGP speakers, any UPDATE message coming from the outside must be spoofed and discarded. A very clever way to achieve that is to set TTL to 255. When a BGP speaker receives an UPDATE message, it checks whether the TTL is 255; if not, the message will be discarded. Since routers will automatically deduct the TTL filed by one, and the largest TTL value that can be set initially in the packet is 255, if a packet passes through one router, its TTL value will be less than 255.

The use of a packet's TTL (for IPv4) or Hop Limit (for IPv6) to verify whether the packet was originated by an adjacent node on a connected link has been used in many recent protocols [Pignataro et al., 2007]. In BGP, this mechanism is referred to as BGP TTL security hack.

By default, BIRD does not use this security mechanism, but it can be enabled by both ends of a BGP peering session. In the following, we enabled this mechanism in the peering between AS-150 and AS-3. Once this is enabled, if we look at the UPDATE messages between these two BGP speakers, we will see that the TTL of the packets becomes 255.

```
// On as150r-router0-10.150.0.254
protocol bgp u_as3 {
    ipv4 {
      ... (omitted) ...
    };
    local    10.100.0.150 as 150;
    neighbor 10.100.0.3 as 3;
    ttl security;
}

// On as3r-r100-10.100.0.3
protocol bgp c_as150 {
    ipv4 {
      ... (omitted) ...
    };
    local    10.100.0.3 as 3;
    neighbor 10.100.0.150 as 150;
    ttl security;
}
```

27.8 Path Selection

BGP routers typically receive multiple routes to the same network. It needs to select one to forward to its peers, as well as for its own routing. Let us check the route to `10.152.0.0/24` from AS-150's BGP router. We can see two entries, each with a different AS path, which lists all the ASes that need to be traversed to reach the location where the network is advertised from. The example shows that there are two different ways to reach `10.152.0.0/24` (a network in AS-152), one via AS-2 and the other via AS-3.

```
# birdc show route all 10.152.0.0/24
10.152.0.0/24        unicast [u_as2 21:32:21.862] * (100) [AS152i]
        via 10.100.0.2 on ix100
        Type: BGP univ
        BGP.origin: IGP
        BGP.as_path:  2 12 152
        BGP.next_hop: 10.100.0.2
        BGP.local_pref: 10

                     unicast [u_as3 21:32:21.625] (100) [AS152i]
        via 10.100.0.3 on ix100
        Type: BGP univ
        BGP.origin: IGP
        BGP.as_path:  3 12 152
        BGP.next_hop: 10.100.0.3
        BGP.local_pref: 10
```

All these routes are kept in the BGP routing table, but BGP will run a best route selection algorithm to choose one route as the current best route. This route will be the one announced to the peers; it is also the one given to the kernel routing table, so routing is based on this selected route. In the example, the first path (marked by the $\star$ symbol) is the one selected for the routing. We can verify that by looking at the kernel routing table. We can see that `10.100.0.2` is the next hop, which is the same as the `next_hop` attribute of the first route above.

```
# ip route show 10.152.0.0/24
10.152.0.0/24 via 10.100.0.2 dev ix100 proto bird metric 32
```

If for some reasons, the current best route is retracted, BGP re-runs the best route selection algorithm to find a new best route. Let's do an experiment. We break the peering between AS-2 and AS-12 at IX-101. This will trigger a series of BGP UPDATE messages. As results, a new best route is selected. This time, the path via AS-3 is selected over the path via AS-2.

```
# birdc show route all 10.152.0.0/24
BIRD 2.0.7 ready.
Table master4:
10.152.0.0/24        unicast [u_as3 06:43:44.590] * (100) [AS152i]
   BGP.as_path: 3 12 152
   BGP.next_hop: 10.100.0.3

                     unicast [u_as2 12:19:54.061] (100) [AS152i]
   BGP.as_path: 2 3 12 152
   BGP.next_hop: 10.100.0.2
```

27.8.1 The Best Path Selection Algorithm

Although different BGP software may implement the path selection differently, they mostly follow the similar order. The following lists the common criteria used by most algorithms. They are arranged according to their priorities.

1. Prefer the path with the highest weight.
2. Prefer the path with the highest local preference.
3. Prefer the path that was locally originated via a network or aggregate BGP subcommand or through redistribution from an IGP.
4. Prefer the path with the shortest AS path.
5. Prefer the path with the lowest origin type (IGP is lower than EGP).
6. Prefer the path with the lowest multi-exit discriminator (MED).
7. Prefer eBGP over iBGP paths.
8. There are more rules to break the ties; they are omitted here.

Because BIRD does not implement weight, so the the local preference is actually the most important criterion, and the length of the AS path is the third most important criterion. We will do some experiments to see how they affect the best path selection.

27.8.2 Local Preference Value

AS-150 peers with both AS-2 and AS-3, which provide the Internet services to AS-150. We call AS-150 multihomed stub AS, as it connects to two providers. The bandwidth provided by AS-2 is more expensive than that from AS-3, so to save money, AS-150 decides to use AS-3 as the primary service provider, while using AS-2 as the backup. Namely, AS-150 wants all the traffic from AS-150 to exit from the AS-3 link, unless this link is broken.

Currently, on AS-150's BGP router, for some destinations, AS-2 is selected, while for others, AS-3 is selected. We can verify that by inspecting the kernel routing table. At IX-100, the BGP router for AS-2 is `10.100.0.2`, and the BGP router for AS-3 is `10.100.0.3`.

```
# ip route
10.152.0.0/24 via 10.100.0.2 dev ix100 proto bird metric 32
10.153.0.0/24 via 10.100.0.2 dev ix100 proto bird metric 32
10.154.0.0/24 via 10.100.0.2 dev ix100 proto bird metric 32
10.155.0.0/24 via 10.100.0.2 dev ix100 proto bird metric 32
10.156.0.0/24 via 10.100.0.2 dev ix100 proto bird metric 32
10.160.0.0/24 via 10.100.0.3 dev ix100 proto bird metric 32
10.161.0.0/24 via 10.100.0.3 dev ix100 proto bird metric 32
10.162.0.0/24 via 10.100.0.3 dev ix100 proto bird metric 32
10.163.0.0/24 via 10.100.0.2 dev ix100 proto bird metric 32
10.164.0.0/24 via 10.100.0.2 dev ix100 proto bird metric 32
...
```

Our goal is to make sure that the AS-3 link is always selected, i.e., for all the external networks, the next hop router should be `10.100.0.3`. We can achieve that using the local preference value, which is an attribute set on the routes received from a peer. In BIRD, this value can be set inside the `import` filter. Originally inside the Emulator, the local preference values for the routes from AS-2 and AS-3 are both set to `10`. We increase the value to `15` for the routes coming from AS-3.

```
# The import filter for the AS-2 peering session
  import filter {
      bgp_large_community.add(PROVIDER_COMM);
      bgp_local_pref = 10;
      accept;
  };

# The import filter for the AS-3 peering session
  import filter {
      bgp_large_community.add(PROVIDER_COMM);
      bgp_local_pref = 15;        ← Change it to 15
      accept;
  };
```

After making the changes, we reload the configuration, and check the routing table again. This time, we can see that the paths via AS-3 are selected for all the destinations, because they have higher local preference values.

```
# birdc configure
Reading configuration from /etc/bird/bird.conf
Reconfigured
```

```
# ip route
10.152.0.0/24 via 10.100.0.3 dev ix100 proto bird metric 32
10.153.0.0/24 via 10.100.0.3 dev ix100 proto bird metric 32
10.154.0.0/24 via 10.100.0.3 dev ix100 proto bird metric 32
10.155.0.0/24 via 10.100.0.3 dev ix100 proto bird metric 32
10.156.0.0/24 via 10.100.0.3 dev ix100 proto bird metric 32
10.160.0.0/24 via 10.100.0.3 dev ix100 proto bird metric 32
10.161.0.0/24 via 10.100.0.3 dev ix100 proto bird metric 32
10.162.0.0/24 via 10.100.0.3 dev ix100 proto bird metric 32
10.163.0.0/24 via 10.100.0.3 dev ix100 proto bird metric 32
10.164.0.0/24 via 10.100.0.3 dev ix100 proto bird metric 32
...
```

Note. The local preference attribute is local to an AS (that is why it is called *local*). Namely, the attribute will not be passed to the peer, so the attribute set by one AS will not affect the path selection of any peer AS. When an AS receives a route from its peer, it sets its own local preference value.

27.8.3 AS Path Prepending

The local preference set by AS-150 only affects the outbound traffic from AS-150, not the inbound, so the packets coming into AS-150 might still go through the more expensive AS-2 link. This is because the local preference value set by an AS does not affect how the peers select their best paths. Let us go to AS-154, and see what path it takes to reach AS-150.

```
# birdc show route all 10.150.0.0/24
10.150.0.0/24        unicast [u_as2 06:43:44.830] * (100) [AS150i]
   via 10.102.0.2 on ix102
   BGP.as_path: 2 150
   BGP.next_hop: 10.102.0.2
   BGP.local_pref: 10

                     unicast [u_as4 06:43:44.830] (100) [AS150i]
   via 10.102.0.4 on ix102
   BGP.as_path: 4 2 150
   BGP.next_hop: 10.102.0.4
   BGP.local_pref: 10

                     unicast [u_as11 06:43:44.830] (100) [AS150i]
   via 10.102.0.11 on ix102
   BGP.as_path: 11 2 150
   BGP.next_hop: 10.102.0.11
   BGP.local_pref: 10
```

We can see that the route with the AS path `"2 150"` is selected as the best path. Therefore, from AS-154, the packets to `10.150.0.0/24` will be given to the AS-2 autonomous system, and eventually reach AS-150 from AS-2 link, the more expensive one. How do we tell the rest of the Internet not to take that path?

AS path prepending is a technique for influencing inbound routing to an AS. The idea is to artificially increase the length of the AS path when announcing a route to a particular peer, by prepending its own autonomous system number (once or several times) to the AS path. According to the path selection algorithm, if the local preference values are the same, the route with the shortest AS path will be selected.

The AS path prepending can be conducted inside the `export` filter, i.e., when a route is exported to the peer. In AS-150's BGP configuration, the original `export` filter for the AS-2 peer is a `where` condition, which only exports the routes that satisfy a condition. To add the AS path prepending to the filter, we change the filter to the following:

```
protocol bgp u_as2 {
 ipv4 {
     table t_bgp;
     ...
     export filter {
        if bgp_large_community ~ [LOCAL_COMM, CUSTOMER_COMM] then {
           bgp_path.prepend(150);
           bgp_path.prepend(150);
           accept;
        }
        reject;
     };
 };
 ..
}
```

After reloading the configuration on AS-150, we go to the AS-154's BGP router, and check the BGP routing table. We can see that now the best path is through AS-4, AS-3, before reaching AS-150, so the inbound traffic to AS-150 will come from the AS-3 link. The path via the AS-2 link is listed as the third route. We can see that due to the two instances of `150` prepended to the AS path, the length becomes four. Given that the local preference values of all these three routes at AS-154 are the same, the length of the AS path becomes the deciding factor.

```
# birdc show route all 10.150.0.0/24
BIRD 2.0.7 ready.
Table master4:
10.150.0.0/24        unicast [u_as4 13:55:42.780] * (100) [AS150i]
        BGP.as_path: 4 3 150
        BGP.local_pref: 10

                     unicast [u_as11 13:55:42.778] (100) [AS150i]
        BGP.as_path: 11 3 150
        BGP.local_pref: 10

                     unicast [u_as2 13:55:42.778] (100) [AS150i]
        BGP.as_path: 2 150 150 150  ← AS path prepending
        BGP.local_pref: 10
```

27.9 BGP Large Communities

When a BGP router sends routes to its peers, they do not send all the routes they know. What routes are sent depends on many factors, such as the region of the peers, the business relationship between the peers, and policies. To help BGP routers make such decisions, additional information needs to be attached to each route, because the predefined set of route attributes cannot capture such information. The BGP communities are created to serve this goal.

BGP communities are attribute tags that can be applied to incoming or outgoing prefixes to achieve some common goal [Li et al., 1996]. For example, ISP can label the incoming routes from a particular autonomous system with a special attribute value, so when it forwards the route to its peers, it can choose to forward the routes only to a particular set of peers (hence the community). The BGP community attribute makes it convenient to enforce such a policy.

The original BGP communities standard was originally proposed in RFC1997 [Li et al., 1996], but it has limitations. That is why the BGP Large communities standard was proposed [Snijders et al., 2017]. BGP Large Communities are composed of three 4-byte integers. Using the canonical notation, this format can be summarized as "ASN:Function:Parameter".

- The first integer is the Global Administrator field, whose value is the Autonomous System Number (ASN) of the AS that has defined the meaning of the remaining two 4-octet fields. Since ASN is unique, each community is unique globally.
- The second integer is the function identifier. This field defines the meaning of the community. Each AS can define their own functions.
- The third integer is the parameter value.

There are many applications of the BGP Large Communities, such as identifying routes by their geographically locations (countries, continent etc.), the business relationships between peers (customers, providers, or peers), and many other aspects. When exporting and importing routes, actions can be applied to these routes based on which communities they belong to. RFC 8195 provides many examples [Snijders et al., 2017].

27.9.1 Communities Defined in the Emulator

In our Internet emulator, we use the BGP large communities to capture the business relationships among peers. These relationships will affect how a route is exported to peers. Let's use AS-150 as an example to show how the BGP large communities are used in our setup. In this autonomous system, we defined four communities.

```
define LOCAL_COMM    = (150, 0, 0);  # large community value: 150:0:0
define CUSTOMER_COMM = (150, 1, 0);  # large community value: 150:1:0
define PEER_COMM     = (150, 2, 0);  # large community value: 150:2:0
define PROVIDER_COMM = (150, 3, 0);  # large community value: 150:3:0
```

When we import or export a route, we can add communities to a route, or delete communities from it. We can also check whether a route belongs to a community or not. These can be done inside the export and import filters. We show a few examples in the following.

```
# Check whether the route belongs to a community
if ((150, 1, 0) ~ bgp_large_community) then return true;
```

```
# Delete community from the route
bgp_large_community.delete([(150, *, *)]);
bgp_large_community.delete([(150, 2, 0)]);

# Add community to the route
bgp_large_community.add((150, 0, 0));
bgp_large_community.add([(150, 1, 0), (150, 2, 0)]);
```

27.9.2 The Local Community

As we have discussed earlier, in the BGP setup, the `direct` protocol is used to generate the routes to the local networks. These routes are put inside the `t_direct` table first, and then they will be exported to the `t_bgp` table (containing all the BGP routes) through a pipe. A filter is applied during the exporting, so the routes are tagged with the `LOCAL_COMM` community, indicating that the routes are locally generated.

```
protocol pipe {
    table t_direct;
    peer table t_bgp;
    import none;
    export filter { bgp_large_community.add(LOCAL_COMM); ... };
}
```

If we use the `static` protocol to provide pre-defined route, we put them in the `LOCAL_COMM` community, indicating that they are locally generated.

```
protocol static {
    ipv4 {
        table t_bgp;
    };
    route 10.150.1.0/24 via 10.150.0.254 {
        bgp_large_community.add(LOCAL_COMM);
    };
    route 10.150.2.0/24 blackhole {
        bgp_large_community.add(LOCAL_COMM);
    };
}
```

27.9.3 The Provider and Customer Communities

Let us look at the peering between AS-150 and AS-2. AS-2 is a transit AS, which is the Internet service provider to AS-150. In the real world, AS-150 will pay AS-2 to get the service. Therefore, in AS-150's BGP configuration, the routes from AS-2 are put in the `PROVIDER_COMM` community, indicating that the routes are from the provider. Correspondingly, in AS-2's BGP configuration, all the routes from AS-150 are put in the `CUSTOMER_COMM` community. The following is AS-150's configuration.

```
protocol bgp u_as2 {
  ipv4 {
    table t_bgp;
    import filter {
      bgp_large_community.add(PROVIDER_COMM);
      ...
    };
    export where bgp_large_community ~ [LOCAL_COMM, CUSTOMER_COMM];
  };
  ...
}
```

Using the community information, an autonomous system can now enforce the policies corresponding to its relationship with peers. As a provider to AS-150, AS-2 is obligated to serve as the transit for AS-150's networks and its customer's networks (in our setup, AS-150 is a stub AS, so it does not have customers). This policy is enforced from the export filter of AS-150. We can see that AS-150 only exports the routes from the `LOCAL_COMM` and `CUSTOMER_COMM` communities to AS-2. This means, only the routes locally generated by or the routes from AS-150's customers are sent to AS-2. Routes in other communities (such as the `PEER_COMM` community discussed below) will not be sent to AS-2.

27.9.4 The Peer Community

AS-150 also peers with another stub autonomous system AS-151, but they are not in a provider-to-customer peering relationship. They peer with each other so the traffic between them can go to each other directly, instead of going through another autonomous system. This is for mutual benefit, so typically they do not pay each other.

Because of this kind of business relationship, they do not want to become a provider to the other. How they forward the routes to each other will be based on the peer-to-peer policy, not on the provider-to-customer policy. Let's see how such a policy can be enforced using the BGP large communities. Here is the peering configuration between AS-150 and AS-151:

```
protocol bgp p_as151 {
  ipv4 {
    table t_bgp;
    import filter {
        # Put the route in the PEER_COMM community
        bgp_large_community.add(PEER_COMM);
        ...
    };
    # Only export the routes from the specified communities
    export where bgp_large_community ~ [LOCAL_COMM, CUSTOMER_COMM];
  };
  ...
}
```

When two autonomous systems are in the peer-to-peer relationship, none of them will serve as the transit for the others in either upstream or downstream directions. Here is how this is achieved using the BGP large communities.

- When AS-150 exports routes to AS-151, it only exports the routes from these two communities: `LOCAL_COMM` and `CUSTOMER_COMM`. Namely, it only exports the routes belonging to itself and its customers (it does not have customers in our setup). It will not export the routes from AS-2 to its peer AS-151, because those routes are put in the

PROVIDER_COMM community (see the peering setup with AS-2). Therefore, even though AS-150 can reach the rest of the Internet via AS-2, it does not tell AS-151, so AS-151 will never use AS-150 to reach the Internet even if AS-151 loses its connection with its own Internet service providers. This policy essentially prevents AS-151 from using AS-150 as a upstream transit to reach the Internet.

- When AS-150 import routes from AS-151, it put the routes in the PEER_COMM community. Based on the export filter defined in the peering with AS-2, AS-150 only exports the routes belonging to itself and its customers to AS-2, not the routes in the PEER_COMM community. Therefore, even though AS-150 can reach AS-151, it does not announce that to AS-2, so AS-2 will never route the AS-151 bound packets to AS-150, essentially preventing AS-2 from using AS-150 as a downstream transit to reach AS-151.

In the Emulator, at IX-100, the autonomous systems AS-2, AS-3, and AS-4 peer with one another. These ASes emulate the Tier-1 autonomous systems, and the peering among them are the peer-to-peer type, not the provider-to-customer type, as they will not provide transit services to the others.

27.10 BGP for Transit Autonomous System

So far, we have only looked at stub autonomous systems, which typically has only one BGP router. Another type of AS has multiple BGP routers, located in different Internet exchanges, where it peers with other ASes. Once packets get into its networks, they will be pulled from one IX to another IX (typically via some internal routers), and eventually be handed over to another AS. This type of AS provides the transit service for other ASes. That is how the hosts in one AS can reach the hosts in another AS. This special of AS is called *Transit AS*.

Figure 27.8 shows the diagram of AS-3, a transit AS inside the Emulator. It has four BGP routers, located in four different cities, where they peer with other autonomous systems. Inside AS-3, these four BGP routers are connected via AS-3's internal networks. Traffic entering from one BGP router will be routed towards the BGP router in another city, where the traffic can exit and get on to another autonomous system.

The question is how these BGP routers inside the same AS work together. When a packet going to AS-161 arrives at router A, it should be routed towards router D, because D is the only router that connects to AS-161, but how does A know that? Moreover, how can the packets be routed inside AS-3. These are achieved using the following two protocols. We will study them in the rest of this section.

- All the BGP routers inside AS-3 communicate with one another via the Internal BGP (IBGP) protocol, so they can forward to one another the BGP routes obtained from their external peers.

- All the routers, including the BGP routers and the internal non-BGP routers, run an Interior Gateway Protocol (IGP) protocol, so they know how to reach the networks contained in the route.

27.10.1 Internal BGP (IBGP)

Just like the peering of BGP routers from different autonomous systems, for the BGP routers in the same autonomous systems to exchange information, they also need to peer with each other

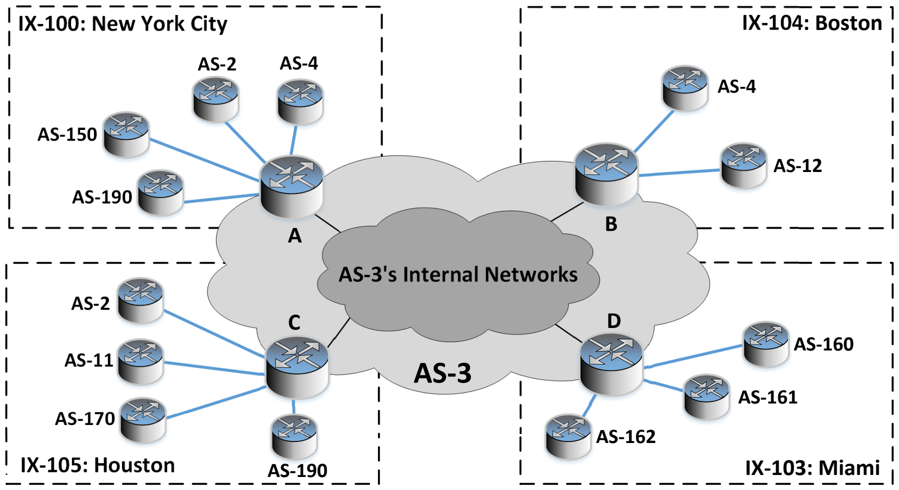

Figure 27.8: A transit AS in the Emulator

and run the BGP protocol to exchange information. The BGP protocol conducted by the BGP routers inside the same AS is called Internal BGP (IBGP), while the BGP protocol conducted by the BGP routers from different ASes is called External BGP (EBGP).

In BIRD, the way to define an IBGP session is the same as defining an EBGP session. When we establish a BGP session between two routers with the same ASN, it will be considered as an IBGP session, and when the session is between two routers with different ASNs, the session is considered as an EBGP session. IBGP and EBGP are basically the same protocol, except for the following different behaviors:

- In IBGP sessions, when sending routes to peers, routers will not prepend their own ASN in the `AS_PATH`, and the `next-hop` attribute will not be altered either.

- RFC 4271 prohibits the advertisement of the routes received from an IBGP peer to another IBGP peer; otherwise, there will be a loop. This is because IBGP does not add their own information to the AS path, so BGP routers will not be able to know whether their peers already know the AS path or not. If forwarding is enabled, a BGP router will keep forwarding information to its peers, creating an endless loop. Because there is no forwarding, RFC 4271 states that all the BGP routers within a single AS must be fully meshed, i.e., all BGP routers peer with one another internally.

  This "full mesh" requirement does not scale well when there are a number of IBGP speakers. An alternative is to use a route reflector [Bates et al., 2000], which is a special type of IBGP speaker. Unlike other IBGP speakers, a route reflector is allowed to advertise an IBGP learned route to another IBGP peer. Using this approach, IBGP routers can simply peer with a route reflector, through which, they can now receive the route advertised from the other IBGP speakers. There is no need for them to be fully meshed. Route reflector is used more in the real world since it simplifies management and configuration.

- EBGP peers are typically directly connected, while IBGP peers are usually multiple hops away. Therefore, IBGP peers rely on the internal routing to reach one another.

AS-3 has four BGP routers, so each of them must peer with the other three. Let us take a look at AS-3's BGP router at IX-103 (Miami). The following is its IBGP configuration.

```
protocol bgp ibgp1 {
    ipv4 {
        table t_bgp;
        import all;
        export all;
        igp table t_ospf;
    };
    local    10.0.0.7 as 3;
    neighbor 10.0.0.6 as 3;    ← IBGP peer
}
protocol bgp ibgp2 {
    ipv4 {
        ... same is ibgp1 ...
    };
    local    10.0.0.7 as 3;
    neighbor 10.0.0.5 as 3;    ← IBGP peer
}
protocol bgp ibgp3 {
    ipv4 {
        ... same is ibgp1 ...
    };
    local    10.0.0.7 as 3;
    neighbor 10.0.0.8 as 3;    ← IBGP peer
}
```

Loopback interface. A BGP router has multiple IP address (one for each network interface), so which IP address should be used in the IBGP peering? Any of the IP addresses will work, but if we use the IP address of a particular network interface, if that interface goes down or gets disabled, the IP address become unreachable, and the peering using the IP address will fail (even though the peers can still reach each other via another path).

To solve this problem, it is suggested that a loopback interface is used in the IBGP peering. The loopback interface is virtual and always stays up. Therefore, the IGBP session can still remain intact even if some other interfaces fail. In the event of link failure, the interior routing protocol will automatically find an alternate path to the peer's loopback IP address.

In the configuration, each of the `10.0.0.x/32` addresses is the IP address of the loopback interface (called `dummy` in our setup). These addresses are announced via the OSPF routing protocol, so all the routers inside the same AS know how to reach these addresses.

The `NEXT_HOP` attribute. When a BGP router sends a route to an internal BGP peer that is multiple hops away, the `NEXT_HOP` route attribute makes no sense to the peer, because this next hop (assuming it is `X`) is the addresses of a boundary routers directly connected to the sender, not the receiver. The peer must figure out what the immediate next hop is in order to reach `X`, as well as finding the distance to `X` (for path selection purpose). To achieve that, after receiving a route from an internal BGP peer, the BGP router will do a route lookup using a routing table. This routing table is an IGP routing table containing the AS-internal routes. The "`igp table`" entry specifies this IGP routing table. IGP will be discussed a little bit later.

27.10.2 Experimenting with IBGP in AS-3

Let us go to AS-3's BGP router at IX-103, and show the route to `10.150.0.0/24`, which is AS-150's network. AS-150 peers with AS-3 at IX-100, not at IX-103, but due to IBGP, AS-3's BGP router at IX-103 now knows how to reach AS-150. See the followings:

```
# birdc show route all 10.150.0.0/24
BIRD 2.0.7 ready.
Table master4:
10.150.0.0/24    unicast [ibgp1 ...] * (100/20) [AS150i]
    via 10.3.0.254 on net_100_103
    Type: BGP univ
    BGP.origin: IGP
    BGP.as_path: 150
    BGP.next_hop: 10.100.0.150
    BGP.local_pref: 30
    BGP.large_community: (3, 1, 0) (150, 0, 0)
```

We can see that the `next_hop` router is still `10.100.0.150`, which is AS-150's BGP router at IX-100. Obviously, the AS-3's BGP router at IX-103 is not connected to this next-hop router, so it conducts a route lookup using its routing table, and find out that to reach `10.100.0.150`, the actual next-hop router should be `10.3.0.254`. That is why when we look at the kernel routing table, we do see the correct next-hop router information.

```
# ip route show 10.150.0.0/24
10.150.0.0/24 via 10.3.0.254 dev net_100_103 proto bird metric 32
```

Disabling IBGP. Let us go to AS-3's BGP router at IX-100, disable one of the IBGP sessions, and see what happens. We run the BIRD client program `birdc` first, and then disable `ibgp1`, which is the IBGP peering between the router at IX-100 and the router at IX-103.

```
# birdc
bird> show protocols
Name       Proto      Table      State  Since         Info
...
ibgp1      BGP        ---        up     20:19:03.800  Established
ibgp2      BGP        ---        up     20:19:11.921  Established
ibgp3      BGP        ---        up     20:20:50.238  Established

bird> disable ibgp1
bird> show protocols ibgp1
Name       Proto      Table      State  Since         Info
ibgp1      BGP        ---        down   20:26:44.526
```

Let us go to AS-3's BGP router at IX-103, and check how to reach AS-150's network `10.150.0.0/24`. Since AS-150 peers with AS-3 at IX-100, but inside AS-3, the IBGP session between IX-100 and IX-103 is disabled, the BGP router at IX-103 does not know how to reach AS-150. That is why we get the following results, i.e., AS-150 is not reachable. To enable the IBGP session again, we can use "`enable ibgp1`".

```
# birdc show route all 10.150.0.0/24
```

```
BIRD 2.0.7 ready.
Network not found

# ip route show 10.150.0.0/24
#   ← No entry in the kernel routing table

# ping 10.150.0.71
ping: connect: Network is unreachable
```

27.10.3 Interior Gateway Protocol (IGP)

Routers inside an autonomous system need to communicate with each other, so they can tell each other what networks they are connected to, and every one can figure out the best path to get to those networks. This is the purpose of the Interior Gateway Protocol (IGP). There are several IGP protocols, including OSPF (a link state routing protocol) and RIP (a distance-vector routing protocol). BIRD supports both of them, but the Emulator only uses OSPF.

Detailed configuration of OSPF can be quite complicated, and it is beyond the scope of this lab. We will study how the BGP routers in the Emulator configures OSPF. More specifically, we will look at the OSPF configuration on AS-3's BGP router at IX-100. First, let us list all the network interfaces on this router:

```
# ip address
1: lo:
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
2: dummy0:
    link/ether 8a:90:16:86:29:9f brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.5/32 scope global dummy0
5311: net_100_103@if5312:
    link/ether 02:42:0a:80:03:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.3.0.254/24 scope global net_100_103
5319: net_100_105@if5320:
    link/ether 02:42:0a:80:04:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.3.1.254/24 scope global net_100_105
5321: ix100@if5322:
    link/ether 02:42:0a:80:1e:03 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.100.0.3/24 scope global ix100
```

The following is the OSPF configuration on the BGP router. An area with ID 0 is called a backbone area. All other areas need to be connected to the backbone area. In our configuration, we put all routes in the backbone area to make things simple.

```
protocol ospf ospf1 {
    ipv4 {
        table t_ospf;  # Store the routes in this table
        import all;
        export all;
    };
    area 0 {
        interface "net_100_103" { hello 1; dead count 2; };
        interface "net_100_105" { hello 1; dead count 2; };
        interface "dummy0" { stub; };
        interface "ix100"  { stub; };
    };
}
```

Other than the `lo` interface, all the network interfaces on the BGP router are included in the backbone area, so routes to the networks attached to these interfaces will be generated and disseminated via the OSPF protocol.

- The `net_100_103` and `net_100_105` interfaces connect the BGP router to the internal networks. These networks should participate in the OSPF protocol. Namely, routes should be generated for these two networks, and the BGP router will exchange route information with the routers inside these networks.

  The option "`hello 1`" means sending periodic hello messages every 1 second; "`dead count 2`" means when the router does not receive any messages from a neighbor in 2*1 seconds, it will consider the neighbor down.

- The IP address of the `dummy0` interface (a loopback interface) is the one used in IBGP, so it must participate in the OSPF protocol; otherwise, the peers will not know how to reach the IP address. The `stub` means that the information from this network will be used in the OSPF protocol, but we will not run OSPF on this network (this network has only one host).

- The `ix100` interface connects AS-3's BGP router to the external network provided by IX-100 for the peering purpose. We do need to include this network in the OSPF protocol, so the internal router knows how to reach this network. The `stub` option indicates that the router will not run the OSPF protocol with anybody on this external network. This is because the hosts on this network do not belong to AS-3; they are outsiders. We do not want to leak the internal network information to the outside, nor do you want the outsider to manipulate your internal routes using OSPF. We should only run OSPF with the routers inside the same autonomous system.

27.10.4 Experimenting with IGP in AS-3

Let us do some experiments with IGP inside AS-3. We will disable the OSPF routing protocol, and see how it affects the routing. Using the following commands, we disable the OSPF protocol on AS-3's BGP router at IX-100. We refer to this BGP router as X.

```
# bridc
birdc> show protocols
...
ospf1      OSPF       t_ospf     up     19:49:43.343  Running
...

birdc> disable ospf1
birdc> show protocols ospf1
ospf1      OSPF       t_ospf     down   19:57:37.187
```

After disabling the OSPF, we check the kernel routing table on X. We see the following results (only the unreachable destinations are listed; the others are omitted).

```
# ip route
unreachable 10.3.2.0/24 proto bird metric 32     ← Internal network
unreachable 10.3.3.0/24 proto bird metric 32     ← Internal network
unreachable 10.160.0.0/24 proto bird metric 32   ← AS-160's network
unreachable 10.161.0.0/24 proto bird metric 32   ← AS-161's network
unreachable 10.162.0.0/24 proto bird metric 32   ← AS-162's network
unreachable 10.170.0.0/24 proto bird metric 32   ← AS-170's network
```

Without OSPF, router X does not know how to reach the other two internal networks in AS-3, as X is not directly connected to them. Moreover, X does not know how to reach the edge routers of AS-160, AS-161, AS-162, and AS-170, as they are connected to the BGP routers at other locations. Without OSPF, those BGP routers cannot share their routes to those edge routers with X, so X will not be able to learn how to reach them.

The experiment clearly shows that for an AS with multiple BGP routers at different locations, an interior gateway protocol is essential. When we specify the IBGP protocol, we have to specify an IGP table using the `igp` option, so each BGP router can use the table to find out the route to the next-hop routers that are multiple hops away.

```
protocol bgp ibgp1 {
    ipv4 {
        table t_bgp;
        import all;
        export all;
        igp table t_ospf;
    };
    local 10.0.0.5 as 3;
    neighbor 10.0.0.6 as 3;
}
```

27.11 IP Anycast: a BGP Application

IP Anycast is a network addressing and routing methodology in which a single IP address is shared by multiple machines (usually in different locations). When we send a packet to this IP address, one of the computers will get the packet. Exactly which one will get it is hard to tell, because it depends on the routing. IP anycast is naturally supported by BGP. A well-known applications of IP anycast is DNS. There are 13 IP addresses for the DNS root servers, but each IP address exists in multiple locations. If we send a DNS query to a root IP address, one of the servers with that IP address will get the query and reply to us. Which one gets the packet depends on the BGP routing.

27.11.1 An Example of IP Anycast

We will use a concrete example to illustrate how IP anycast works. It will be better to use the visualization map that comes with the Emulator. The map is hosted on a web server with this URL: `http://localhost:8080/map.html`. In the Emulator (see Figure 27.9), AS-190 has two Points of Presence (PoP), one at IX-100 (where it peers with AS-4), and the other at IX-105 (where it peers with AS-3). It is quite common for a transit AS to have PoPs at multiple locations, and these PoPs will be connected via the internal networks of the AS. However, AS-190 is just a stub AS, and its two PoPs are disconnected.

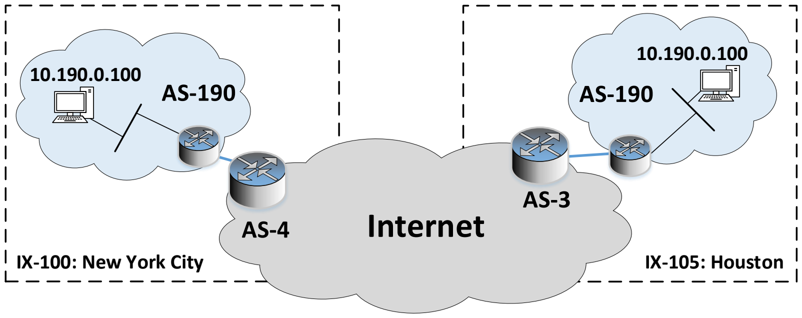

Figure 27.9: AS-190 and IP Anycast

A closer look at the networks at AS-190's two PoPs reveals that these two networks have the identical network prefix `10.190.0.0/24`. Both networks have a host with the identical IP address `10.190.0.100`. If we send a packet to this IP address from two different locations, will they both receive the packet, or will only one of them get it, which one?

Let us do an experiment first. We run "`ping 10.190.0.100`" from two different locations: a host in AS-156 and a host in AS-160. On the map, we set the filter to `icmp` to visualize the ICMP traffic. We can see that each `ping` reaches a different destination, so for each packet, only one of the host with the address `10.190.0.100` gets the packet, not all. This is not a unicast, not a broadcast, but an anycast.

27.11.2 How IP Anycast Works

When the BGP routers of AS-190 peer with other ASes at two different locations, they both announce the network prefix `10.190.0.0/24` to their peers, AS-3 and AS-4, who will further announce the prefix to their peers. Therefore, some ASes, especially well-connected transit ASes, will see more than one AS paths to this network prefix. For example, if we go to AS-11's BGP router at IX-102, and check its BGP routing table, we see three AS paths (the first one is the selected path):

```
# birdc show route all 10.190.0.0/24
BIRD 2.0.7 ready.
Table master4:
10.190.0.0/24
        via 10.102.0.4 on ix102
        BGP.as_path: 4 190                  AS path ①
        BGP.next_hop: 10.102.0.4

        via 10.11.0.253 on net_102_105
        BGP.as_path: 3 190                  AS path ②
        BGP.next_hop: 10.105.0.3

        via 10.102.0.2 on ix102
        BGP.as_path: 2 4 190                AS path ③
        BGP.next_hop: 10.102.0.2
```

Therefore, it does not matter whether AS-190 announces its network prefixes from one location or multiple locations, and whether the network prefixes overlap or not, to BGP, they are just different AS paths. Each BGP router will use the path selection algorithm to pick the best path, and announce it to the peers. Since the length of the AS path is an important criterion in the algorithm, when two AS paths have a tie in the more important criteria (such as the local preference), the path with a shorter AS path will be selected. This in general means that the destination closer to the sender will be the one receiving the packet. Of course, this is not always true, as a shorter AS path does not necessarily mean a shorter distance.

Let us go back to the experiment, and see why the `ping` packets from AS-156 reach a different destination than the ones from AS-160. Let us take a look at their BGP routing tables:

```
-------------------------
From AS-156's BGP router
-------------------------
# birdc show route all 10.190.0.0/24
BIRD 2.0.7 ready.
Table master4:
10.190.0.0/24        unicast [u_as4 12:18:43.600] * (100) [AS190i]
     via 10.102.0.4 on ix102
     Type: BGP univ
     BGP.origin: IGP
     BGP.as_path: 4 190    ← AS-190 peers with AS-4 only at IX-100
     BGP.next_hop: 10.102.0.4
     BGP.local_pref: 10
     BGP.large_community: (4, 1, 0) (190, 0, 0) (156, 3, 0)

-------------------------
From AS-160's BGP router
-------------------------
# birdc show route all 10.190.0.0/24
BIRD 2.0.7 ready.
Table master4:
10.190.0.0/24        unicast [u_as3 12:18:49.109] * (100) [AS190i]
     via 10.103.0.3 on ix103
     Type: BGP univ
     BGP.origin: IGP
     BGP.as_path: 3 190    ← AS-190 peers with AS-3 only at IX-105
     BGP.next_hop: 10.103.0.3
     BGP.local_pref: 10
     BGP.large_community: (3, 1, 0) (190, 0, 0) (160, 3, 0)
```

From the routing table, we can now explain our observations.

- The AS path obtained by AS-156's BGP router is `"4 190"`. Since AS-190 peers with AS-4 only at IX-100, this means packets from AS-156 will be routed towards New York City, where IX-100 is located.

- The AS path obtained by AS-160's BGP router is `"3 190"`. Since AS-190 peers with AS-3 only at IX-105, this means packets from AS-160 will be routed towards Houston, where IX-105 is located.

27.11.3 Applications of IP Anycast

IP anycast is mostly used for the stateless communication, such as UDP. This is because there is no guarantee that the packets to an IP anycast address will always reach the same destination. If routes changes and different paths are selected by BGP, packets may now reach a different destination.

The most well-known application of IP anycast is DNS, which uses the stateless UDP protocol. DNS root servers (A to M) have 13 IP addresses, but they are not just 13 servers. All these addresses use IP anycast to provide distributed services. For example, the F-Root server has the IP address `192.5.5.241`, which belongs to AS-3557. According to `https://root-servers.org/`, the F-Root has a presence at more than 200 locations, peering with over 3000 peers (as of September 2021). By providing presence at many locations around the world, DNS root servers can get closer to the Internet users, which rely on this important Internet infrastructure.

IP anycast can also be used for content delivery networks to deliver static content via HTTP, such as images. Although HTTP uses a stateful TCP protocol, the general stability of routes and statelessness of connections makes anycast also suitable for this type of applications.

27.12 BGP Hijacking Attack

BGP does not have a built-in security mechanism, so there is no mechanism to verify the authenticity of the data exchanged among the peers. There is no efficient way to verify whether a route advertisement or withdrawal is legitimate or spoofed. The most common and severe BGP attack is called BGP hijacking, also known as IP prefix hijacking, prefix hijacking, and IP hijacking. It can hijack a network prefix, causing the traffic to the target prefix to be rerouted, and eventually dropped. This type of attack occurs quite frequently on the Internet. Although most of them are due to router misconfiguration, it does indicate how vulnerable the BGP protocol is.

27.12.1 Routing Rule: Longest Match

Before talking about how the prefix hijacking works, we need to understand routing a little bit more. When two prefixes in the routing table overlaps, and a destination matches both prefixes, which one will be selected?

Let us figure this out using an experiment. We will go to AS-150's host (`10.150.0.71`), ping `10.164.0.71`. Using the map, we can visualize the ICMP traffic, so we can see the packet trace. Now we go to AS-150's BGP router, and check its routing table. We see one entry to the `10.164.0.0/24` network. We add a new routing entry to the `10.164.0.0/25` network, which is a subnet of `10.164.0.0/24`. The destination `10.164.0.71` belongs to both networks, so it matches both routing entries. Which entry will be used?

```
// On AS-150's BGP router
# ip route | grep 10.164
10.164.0.0/24 via 10.100.0.2 dev ix100 proto bird metric 32

// Add a new route
# ip route add 10.164.0.0/25 via 10.100.0.3
# ip route | grep 10.164
10.164.0.0/25 via 10.100.0.3 dev ix100
```

```
10.164.0.0/24 via 10.100.0.2 dev ix100 proto bird metric 32
```

We can see that before the new route is added, the ICMP traffic to `10.164.0.71` goes through the AS-2 autonomous system, because the next-hop router is `10.100.0.2`, which belongs to AS-2. Immediately after the route is added, we can see that the traffic gets re-routed, and it now goes through `10.100.0.3`, which belongs to AS-3.

Although `10.164.0.71` matches with both prefixes in the routing table, one prefix (`10.164.0.0/25`) has 25 bits, while the other (`10.164.0.0/24`) has 24 bits. Routers choose the longer match, i.e., the more specific route is selected.

From this experiment, we can see if we can add an entry with a more specific prefix, we can affect router's decisions. The experiments only shows that the routing is partially affected, because the final destination is still the same. This is because we have only affected one router. If we can add such a routing entry to many BGP routers, we may be able to completely change the routing path. That is exactly the objective of of the BGP prefix hijacking attack.

27.12.2 IP Prefix Hijacking

We will use AS-164 as our attack target. We know that this autonomous system has advertised the prefix `10.164.0.0/24` (P) to the entire Internet, so every BGP router has an entry in its routing table for this destination. The prefixes `10.164.0.0/25` (A) and `10.164.0.128/25` (B) are two subnets of P, but jointly, A and B cover the entire address space of P. This can be seen from the following where we use the binary notation to represent the last octet in the IP prefix (* means it can be 0 or 1).

```
10.164.0.0/24   covers 10.164.0.********   ← The target prefix
10.164.0.0/25   covers 10.164.0.0*******
10.164.0.128/25 covers 10.164.0.1*******
```

If A and B are also in the routing table along with P, any IP address in the `10.164.0.0/24` address space will match either A or B, as well as P, but A or B will be picked over P, because the length of their prefixes has 25 bits, longer than P's 24 bits.

The question is how to get A and B into every BGP router. That is quite simple. Attackers just need to advertise A and B to the entire Internet from a BGP router, telling everybody that their autonomous system is the origin of these two prefixes, so packets to these networks should be routed towards them.

Let us give it a try on the Emulator. We choose AS-150 as the malicious autonomous system, and choose AS-164 as the target. Our job is to hijack the `10.164.0.0/24` network prefix owned by AS-164. We add the following entry to the BIRD configuration file on AS-150's BGP router. We need to run "`birdc configure`" to load the updated configuration file to the BIRD daemon.

```
protocol static hijacks {
  ipv4 { table t_bgp; };
  route 10.164.0.0/25 blackhole   {
           bgp_large_community.add(LOCAL_COMM);
  };
  route 10.164.0.128/25 blackhole {
           bgp_large_community.add(LOCAL_COMM);
  };
}
```

This will add two static routes to the BGP routing table. The action `blackhole` means that once a packet to these destination reaches this BGP router, it will be dropped, instead of being routed. AS-150's BGP router will advertise these two routes to its peers, and eventually the route advertisement will reach every BGP router on the Internet (Emulator). We can pick a BGP router, and check its BGP routing table and kernel routing table, we will see both of them:

```
// On AS-170
# birdc show route all 10.164.0.0/24
10.164.0.0/24        unicast [u_as3 13:26:41.574] * (100) [AS164i]
        via 10.105.0.3 on ix105
        BGP.as_path: 3 12 164
        BGP.next_hop: 10.105.0.3

# birdc show route all 10.164.0.0/25
10.164.0.0/25        unicast [u_as3 13:37:00.597] * (100) [AS150i]
        via 10.105.0.3 on ix105
        BGP.as_path: 3 150
        BGP.next_hop: 10.105.0.3

# birdc show route all 10.164.0.128/25
10.164.0.128/25      unicast [u_as3 13:37:00.597] * (100) [AS150i]
        via 10.105.0.3 on ix105
        BGP.as_path: 3 150
        BGP.next_hop: 10.105.0.3
```

From the results, we can see that the original AS path of `10.164.0.0/24` is `"3 12 164"`, while the AS paths of the forged ones are `"3 150"`, which leads to the attacker's autonomous system. In all these paths, the next-hop is the same, which is a router (`10.105.0.3`) in AS-3, but, once packets reach this router, they will be routed towards AS-150, instead of the original destination AS-164. Let us take a look at the routing table on this router.

```
// On 10.105.0.3
# ip route | grep 10.164
10.164.0.0/24   via 10.3.2.254 dev net_103_105 proto bird metric 32
10.164.0.0/25   via 10.3.1.254 dev net_100_105 proto bird metric 32
10.164.0.128/25 via 10.3.1.254 dev net_100_105 proto bird metric 32
```

We can see that the router used for the original prefix (the first one) is different from the one used for the forged prefixes (the second and third ones). Since the forged prefixes will always be picked over the original one, packets towards `10.164.0.0/24` will be routed to `10.3.1.254`. If we take a look at the routing table on this router, we will see the following, which clearly indicates that the packets will be routed to `10.100.0.150`, which is the BGP router belonging to AS-150, the attacker's autonomous system. From there, the packets will be dropped due to the `blackhole` action in the configuration.

```
// On 10.3.1.254
# ip route | grep 10.164
10.164.0.0/24   via 10.3.0.253   dev net_100_103 proto bird metric 32
10.164.0.0/25   via 10.100.0.150 dev ix100 proto bird metric 32
10.164.0.128/25 via 10.100.0.150 dev ix100 proto bird metric 32
```

Using the map. The impact of the attack can be visualized using the Emulator map (see § 27.4.1). We set the filter to `"icmp && dst 10.164.0.71"` on the map to visualize packet trace. Before launching the attack, we ping `10.164.0.71` from a host machine in one of the ASes. We can see the packet trace towards the destination. After the attack is launched, we will see that the packets are immediately rerouted to AS-150, and the ping program will no longer get any reply back. The prefix `10.164.0.0/24` is completely hijacked.

27.12.3 Fighting Back

On Sunday, 24 February 2008, after receiving a censorship order from the government to block YouTube (due to some materials posted on YouTube), Pakistan Telecom (AS17557) decides to block the access to YouTube's IP address, which includes several addresses in the `208.65.153.0/24` network space. The technique used in the blocking is the same as the IP prefix hijacking, i.e., AS17557 started to announce the prefix `208.65.153.0/24` to its peers.

These BGP announcements (BGP UPDATE messages) were supposed to be advertised only to the peers within Pakistan, but a mistake was made, so the UPDATE messages were advertised to one of the upstream peer, the Hong Kong-based PCCW Global (AS3491). AS3491 were supposed to catch this fake announcement, but it failed to do so, and forwarded the fake announcement to the rest of the Internet. This resulted in the hijacking of YouTube traffic on a global scale [RIPE NCC, 2008].

One of the prefixes announced by YouTube (AS36561) is `208.65.152.0/22`. Since `208.65.153.0/24` is a more specific prefix inside the `208.65.152.0/22` address space, packets going to `208.65.153.0/24` will be routed to Pakistan, instead of to YouTube. Therefore, the prefix announced by Pakistan Telecom hijacked part of the YouTube's prefix.

YouTube soon detected this, while trying to resolve the problem in the normal channel (contacting PCCW to correct the mistake), YouTube tried to reclaim its IP prefixes by announcing the same `208.65.153.0/24`, but this did not completely solve the problem, because this prefix has the same length as the one advertised by Pakistan, so which path is picked is up to the path selection algorithm of each BGP router (the length of the AS path is one of the criteria). With this announcement, YouTube could get some of the traffic back, but not all.

YouTube corrected that by announcing two more prefixes: `208.65.153.128/25` and `208.65.153.0/25`. These prefixes covers the entire `208.65.153.0/24` space and they are longer, so all the routers on the Internet started to route the traffic back to YouTube. Eventually, YouTube's contact with PCCW went through, PCCW withdrew the fake announcements, and the problem was fixed.

Helping AS-164 fight back. We can emulate what YouTube did and help AS-164 to reclaim its network back during the attack. For each of the prefix advertised by the attacker, we will create two prefixes that are one bit longer than the one from the attacker. See the following:

```
10.164.0.0/25    covers 10.164.0.0*******  ← By attacker
10.164.0.0/26    covers 10.164.0.00******
10.164.0.64/26   covers 10.164.0.01******

10.164.0.128/25  covers 10.164.0.1*******  ← By attacker
10.164.0.128/26  covers 10.164.0.10******
10.164.0.192/26  covers 10.164.0.11******
```

We add these four prefixes to AS-164's BGP configuration file using the `static` protocol. The interface `net0` is the one used by the BGP router to connect to AS-164's internal network.

```
protocol static {
  ipv4 { table t_bgp; };
  route 10.164.0.0/26 via "net0" {
           bgp_large_community.add(LOCAL_COMM);
  };
  route 10.164.0.64/26 via "net0" {
           bgp_large_community.add(LOCAL_COMM);
  };
  route 10.164.0.128/26 via "net0" {
           bgp_large_community.add(LOCAL_COMM);
  };
  route 10.164.0.192/26 via "net0" {
           bgp_large_community.add(LOCAL_COMM);
  };
}
```

After reloading the configuration, and wait for a few seconds, we can see that the ping program will now get responses, indicating that the packets are now reaching the real destination `10.164.0.71`. We get our traffic back. If we go to any BGP router, we can see the following routing entries:

```
# ip route | grep 10.164
10.164.0.0/24 via 10.102.0.2   ...  ← The original route
10.164.0.0/25 via 10.102.0.2   ...  ← From the attacker
10.164.0.0/26 via 10.102.0.2   ...  ← Fighting back
10.164.0.64/26 via 10.102.0.2  ...  ← Fighting back
10.164.0.128/25 via 10.102.0.2 ...  ← From the attacker
10.164.0.128/26 via 10.102.0.2 ...  ← Fighting back
10.164.0.192/26 via 10.102.0.2 ...  ← Fighting back
```

27.12.4 Filtering Out Spoofed Advertisement

In the YouTube incident, the problem was eventually resolved when PCCW, the upstream service provider for Pakistan Telecom, withdrew the fake announcements. To emulate that, we can add a filter rule to AS-2's and AS-3's configuration (at IX-100, where they peer with AS-150), so when they import routes from AS-150, they only import the route to prefix `10.150.0.0/24`. By doing so, the fake routes announced by AS-150 will not be accepted by AS-2 or AS-3; therefore, they will not be able to reach the Internet.

```
protocol bgp c_as150 {
  ipv4 {
    table t_bgp;
    import filter {
        bgp_large_community.add(CUSTOMER_COMM);
        bgp_local_pref = 30;
        if (net != 10.150.0.0/24) then reject;  ← The added rule
        accept;
    };
    export all;
    next hop self;
  };
  local 10.100.0.3 as 3;
  neighbor 10.100.0.150 as 150;
}
```

27.12.5 Defending Against IP Prefix Hijacking

Defending against IP prefix hijacking is quite challenging. One defense is to use the filtering. As we have discussed in § 27.6.2, BGP speakers can conduct ingress and egress filtering. When an ISP's BGP speaker receives a prefix announcement from its peer, it can check whether the route is indeed owned by the peer. ISP can get the owner information for a prefix from the Internet Routing Registry (IRR). How effective this mechanism is depends on whether the information from IRR is complete or not, and whether the ISP is conducting the filtering correctly. To make things worse, the topology of the ASes is quite complex and some peering relationship is private, making route verification much more complicated, if possible at all [Nordström and Dovrolis, 2004].

Instead of using a central database like IRR for route verification, we can also use cryptography to ensure that a route's authenticity. The can be done via Resource Public Key Infrastructure (RPKI), also known as Resource Certification. RPKI is a specialized public key infrastructure (PKI) framework. It enables an entity to verifiably assert that it is the legitimate holder of a set of IP addresses or a set of Autonomous System (AS) numbers [Lepinski and Kent, 2012].

The details of RPKI are beyond the scope of this chapter. Its specification is documented in a series of RFCs, RFC 6481, 6482, ..., to 6495. An open-source document project on RPKI can be found from `https://rpki.readthedocs.io`. It provides detailed documentation on RPKI. We recommend readers to get more information on RPKI from this site.

27.13 Summary

The Internet consists of many autonomous systems. Internally each AS manages its own networks, but externally, ASes have to work together to advertise and forward route information, so each AS knows where to route packets for any given destination. This is done through BGP. In this chapter, we have conducted an in-depth study of BGP, which is quite complicated. With the help of our SEED Internet Emulator, we are able to look at how BGP works and conduct experiments on BGP in a real system. It allows us to connect the abstract BGP concepts with a real-world experience. Based on the understanding of BGP, we have discussed the BGP hijacking attack. We are able to conduct the actual attack inside the emulator.

❐ Hands-on Lab Exercise

We have developed a SEED lab for this chapter. The lab is called *BGP Exploration and Attack Lab*, and it is hosted on the SEED website: `https://seedsecuritylabs.org`.

❐ Problems and Resources

The homework problems, slides, and source code for this chapter can be downloaded from the book's website: `https://www.handsonsecurity.net/`.

Bibliography

Bates, T. J., Chandra, R., and Chen, E. (2000). BGP Route Reflection - An Alternative to Full Mesh IBGP. RFC 2796.

CZ.NIC (2021). BIRD Internet Routing Daemon. `https://bird.network.cz/`.

Hawkinson, J. A. and Bates, T. J. (1996). Guidelines for creation, selection, and registration of an Autonomous System (AS). RFC 1930.

Janardhan, S. (2021). More details about the October 4 outage. `https://engineering.fb.com/2021/10/05/networking-traffic/outage-details/`.

Lepinski, M. and Kent, S. (2012). An Infrastructure to Support Secure Internet Routing. RFC 6480.

Li, T., Chandra, R., and Traina, P. S. (1996). BGP Communities Attribute. RFC 1997.

Nordström, O. and Dovrolis, C. (2004). Beware of BGP attacks. *Computer Communication Review*, 34:1–8.

Pignataro, C., Savola, P., Meyer, D., Gill, V., and Heasley, J. (2007). The Generalized TTL Security Mechanism (GTSM). RFC 5082.

RIPE NCC (2008). YouTube Hijacking: A RIPE NCC RIS case study. `https://www.ripe.net/publications/news/industry-developments/youtube-hijacking-a-ripe-ncc-ris-case-study`.

Snijders, J., Heasley, J., and Schmidt, M. (2017). Use of BGP Large Communities. RFC 8195.