BGP 实验: 建立一个互联网仿真器

1 概要

边界网关协议（BGP）是用于自治系统（AS）之间交换路由和可达性信息的标准外部网关协议。它是互联网的“粘合剂”，也是互联网基础设施的重要组成部分。同时，它也是一个主要的攻击目标，因为如果攻击者能够控制 BGP，他们可以断开互联网并重定向流量。

由于 BGP 的复杂性，我们很难在一个实验中涵盖所有内容。因此，我们开发了一系列与 BGP 相关的实验。这个实验是该系列的第一个，也是其他所有 BGP 实验的基础。本实验的目标是帮助学生理解 BGP 如何“粘合”互联网以及实际的互联网是如何连接的。在这个实验中，我们将引导学生们构建一个小型规模的互联网。我们称这个互联网为互联网仿真器（或简称仿真器）。这个仿真器将成为后续所有 BGP 实验的基础，同时也用于一些依赖于互联网而非 BGP 的实验。

在任务 1 到 3 中，我们将逐步展示如何构建这个互联网仿真器。大多数文件已经在这些任务中给出。一旦学生理解了构建过程，在任务 4 中，他们将被要求通过增加更多的自治系统、BGP 路由器和互联网交换点来扩展此仿真器。本实验涵盖了以下主题：

- BGP 协议的工作原理
- BGP 配置
- 路由
- 互联网交换点（IXP）

视频 有关 BGP 协议的详细介绍可以在 SEED Lecture at Udemy, Internet Security 的第 10 节找到，见 *Internet Security: A Hands-on Approach*, by Wenliang Du. 详情请见 `https://www.handsonsecurity.net/video.html`. 。该讲课是在开发这个实验之前录制的；它主要关注理论部分，即解释 BGP 协议是如何工作的。此实验提供了实践部分。

实验环境 本实验在我们预先构建好的 Ubuntu 20.04 VM（可以从我们的 SEED 网站当中下载）当中测试可行。既然我们使用容器来建立实验环境，本实验不太依赖 SEED VM。您可以在其他 VM、物理机器以及云端 VM 上进行此实验。

教师自定义 需要注意的是，任务 4 要求教师进行随机选择（以防止抄袭）。

致谢 本实验是在纽约雪城大学电气工程与计算机科学系的一名研究生 Honghao Zeng 的帮助下开发的。该 SEED 项目得到了美国国家科学基金会的支持。

2 约定

在我们的互联网配置中，我们将使用很多数字。它们分为两类：自治系统号（ASN）和 IP 地址（对于网络和主机都是如此）。在现实世界中，ASN 和 IP 地址之间没有特定的关系，并且这些编号的分配也没有特定的模式。在我们所设定的环境中，为了便于记忆那些数字，我们将遵循一些约定。

自治系统号的分配 我们主要有四个类别的自治系统：残桩自治系统、互联网交换中心、小型转送自治系统和大型转送自治系统。以下是自治系统号分配的惯例：

- `1`：保留。
- `2 - 19`：为大型转送自治系统。
- `20 - 49`：为小型转送自治系统。
- `50 - 99`：保留。
- `100 - 149`：为互联网交换中心。
- `150 - 254`：为残桩自治系统。

分配给自治系统的 IP 前缀 在现实世界中，ASN 和 IP 前缀之间没有关系。而在我们的仿真器中，我们创建了它们之间的人为联系，以便我们可以轻松找到任何给定 ASN 的网络地址。

- 对于自治系统号码 X，其网络前缀为 10.X.0.0/16。
- 如果该自治系统有多重子网，则子网 ID 将为 10.X.1.0/24、10.X.2.0/24 等等。

分配给机器的 IP 地址 当我们将 IP 地址分配给路由器和主机时，我们遵循以下约定。我们只将此约定应用于 IP 地址的最后 8 位。

- `71 - 99`：为普通主机。从 `71` 开始分配。
- `200 - 254`：为 BGP 路由器。从 `254` 反向开始分配。

3 任务 1. 建立自治系统的容器

为了构建互联网仿真器，我们需要运行许多机器，有些充当路由器，有些作为主机。我们将使用 Docker 容器来实现这一点，每个机器作为一个容器运行。在这个任务中，我们将创建两个自治系统 `AS150` 和 `AS151`。在每个自治系统内，我们将会创建一个网络、一个 BGP 路由器和一个普通主机。它们的 IP 地址如下所示。我们将使用容器技术来实现这一点。本任务所需的所有文件可以在 `Base_Images` 和 `Task1_AS` 文件夹中找到。

```
AS150:
Network:     10.150.0.0/24
BGP router:  10.150.0.254
Host:        10.150.0.71

AS151:
```

```
Network:     10.151.0.0/24
BGP router:  10.151.0.254
Host:        10.151.0.71
```

3.1 基础容器镜像

在这个实验中，所有的 Docker 镜像都基于两个基础镜像：一个是主机，另一个是路由器。这两个镜像几乎完全相同，只是主机会附带一个 nginx Web 服务器，而路由器会附带一个 bird BGP 服务器。我们已经构建了这些基础镜像，并使他们在 Docker Hub 中可用。

```
Host:  handsonsecurity/seed-server:nginx
BGP:   handsonsecurity/seed-server:bgp
```

尽管我们可以直接使用 Docker Hub 上的基础镜像来构建容器镜像，但为了减少对 Docker Hub 的访问次数（因为 Docker Hub 公司对一个六小时的窗口可以进行多少次访问有限制），我们首先使用 Docker Hub 中的基础镜像构建本地基础镜像，然后再利用该本地基础镜像构建本实验中所需的其他所有容器镜像。

3.2 主机的容器

以下为用于主机容器的 Dockerfile 内容。它基于上述描述的基础镜像，其中 seed_base_host 是基础镜像的名字，这是在 Docker Compose 文件中给出的名字，我们稍后会进行讨论。在构建镜像时，我们将一个网页文件（index.html）复制到 web 目录中。

```
FROM seed_base_host

COPY index.html /var/www/html/

CMD ip route change default via ${DEFAULT_ROUTER} dev eth0 \
    && service nginx start \
    && tail -f /dev/null
```

CMD 条目指定了容器启动时将执行的命令。我们一共包含了三条命令：第一条命令根据 DEFAULT_ROUTER 环境变量来设置默认路由（该环境变量将在运行容器时进行传递）。第二条命令启动了 nginx Web 服务器。第三条 (tail) 命令将阻止命令结束。如果命令结束，则容器将会立即关闭。

3.3 BGP 路由的容器

BGP 路由器的 Dockerfile 以之前构建的 seed_base_router 基础镜像为基础。它将 bird 配置文件复制到容器中。我们将使用同一个文件夹来构建不同的 Docker 镜像，每个镜像都具有一个不同的 bird 配置文件。所有配置文件都存储在该文件夹内，但要复制哪个文件取决于 BIRD_CONF 参数的值，这个参数将在后面讨论 Compose 文件时进行设置。

```
FROM seed_base_router
```

```
ARG BIRD_CONF

# Copy the bird configuration file
COPY ${BIRD_CONF} /etc/bird/bird.conf

# Delete the default routing entry and start BGP server
CMD ip route del default ; mkdir -p /run/bird && bird -d
```

3.4 Docker Compose

在这个任务中，我们只有四个容器，手动管理并运行这些容器是可行的。然而，随着我们进行后续的任务时，容器的数量将显著增加。对于更复杂的设置，拥有 20 到 30 个容器并不罕见。除了这些容器外，我们还需要创建多个网络，并为每个容器分配 IP 地址。手动管理所有这些内容将会非常困难。

Docker 提供了一个名为 Compose 的工具，用于定义和运行多容器 Docker 应用程序。有了 Compose，我们可以使用 YAML 文件配置我们的容器（如创建网络、为容器分配 IP 地址、配置容器等）。然后只需要一行命令，我们就可以根据我们的配置启动并创建所有容器。

`docker-compose.yml` 文件. Compose 文件有两个主要部分：`services` 部分列出了我们想要构建的所有容器，而 `networks` 部分则列出了我们需要创建的所有网络。以下示例列出了任务中所需的两个网络，而服务条目则被省略。

```
version: "3"

services:
    ... (omitted, will be discussed later) ...

networks:
    as150-net:
        ipam:
            config:
                - subnet: 10.150.0.0/24
        name: seed-as150-net

    as151-net:
        ipam:
            config:
                - subnet: 10.151.0.0/24
        name: seed-as151-net
```

基础镜像 前两个服务条目是我们要首先构建的基础容器镜像。它们是其他容器的基础。`image` 条目指定了镜像的名称。这些名称已经在每个容器 `Dockerfile` 的 `FROM` 条目中给出，因此如果我们在这里更改了名称，则需要更改所有 `Dockerfile` 文件。这两个容器在实验中将不起任何作用，并且它们会在启

动后立即退出。我们仅使用他们的镜像作为其他容器的基础。如我们之前所提到的，我们这样做是为了避免从 Docker Hub 拉取太多的镜像，因为 Docker Hub 为用户设置了限制。

```
seed_base_router:
   build: ../Base_Images/router
   image: seed_base_router
   container_name: seed-base-router
   command: " echo 'exiting ...' " ← 容器立即退出

seed_base_host:
   build: ../Base_Images/host
   image: seed_base_host
   container_name: seed-base-host
   command: " echo 'exiting ...' "  ← 容器立即退出
```

主机容器 我们在每个自治系统中放置一个主机。以下服务条目指定了 AS150 的主机容器。

```
as150_host:
   build: ./host
   image: seed-as-common-host              ← 镜像名称
   container_name: as150-host-10.150.0.71 ← 容器名称
   environment:
         DEFAULT_ROUTER: 10.150.0.254      ← 在 Dockerfile 的 CMD 条目中使用
   cap_add:
         - ALL
   networks:
         as150-net:
           ipv4_address: 10.150.0.71
```

我们为此提供进一步的说明:

- `build: < 文件夹名称 >`: 此条目表示容器的文件夹名称，并将使用该文件夹内的 `Dockerfile` 来构建容器。
- `container_name`: 该条目指定容器的名称。为了方便起见，我们在名称中包含了容器的 IP 地址，这样我们可以通过名称轻松地知道容器的 IP 地址。
- `environment`: 我们指定了一个名为 `DEFAULT_ROUTER` 的环境变量。正如之前所讨论的那样，这个环境变量在 Dockerfile 中的 `CMD` 条目中使用。
- `cap_add`: 该条目为容器分配权限。在我们的设置中，我们将 所有权限分配给容器。这是因为我们需要在容器内进行一些特权操作。容器内的 root 账户并没有与主机机器上的 root 相同的权限。将所有权限授予一个容器会给容器内部的 root 大多数真实 root 的权限。[1]

[1] 容器中的 root 仍然有一些无法做到的事情，例如更改 `sysctl` 参数。

- networks 条目: 它指定了此容器连接到的网络名称以及分配给容器的 IP 地址。你可以将多个网络附加到一个容器上。在任务 2 和 3 中，你会发现在路由器容器上有两个网络（并非这个任务）。

路由器 在本任务中，我们为每个自治系统提供一个 BGP 路由器。以下为 AS150 的 BGP 路由器容器。

```
as150_router:
   build:
       context: ./router
       args:
           BIRD_CONF: as150_bird.conf  ← 在 Dockerfile 中使用
   image: as150-router
   container_name: as150-router-10.150.0.254
   cap_add:
       - ALL
   sysctls:
           - net.ipv4.ip_forward=1     ← 启用 IP 转发
   networks:
         as150-net:
           ipv4_address: 10.150.0.254
```

大部分条目与主机容器的相同。我们在此提供一些进一步的说明:

- args 条目: 在此条目中，我们定义了一个名为 BIRD_CONF 的参数。这个参数在 Dockerfile 中使用，它指定了应该复制到容器镜像中的 bird 配置文件。
- sysctls 条目: 由于此容器将被作为路由器使用，我们需要启用其 IP 转发；否则它将不会转发数据包。

3.5 运行并测试仿真器

我们首先可以使用以下命令构建容器，然后使用 up 命令启动所有容器。如果要停止并删除所有正在运行的容器，我们可以使用 down 命令。我们应当在包含 docker-compose.yml 的同一目录下执行这些命令，因为默认情况下它们会在当前文件夹中寻找配置文件。

```
// 构建容器
$ docker-compose build

// 启动所有容器
$ docker-compose up

// 停止所有容器
$ docker-compose down
```

所有容器都会在后台运行。如果需要在某个容器内执行一些命令，我们只需首先使用 "docker ps" 命令找出该容器的 ID，然后使用 "docker exec" 命令在容器内部运行一个 bash Shell。如果我们使用

-it 选项，则会得到一个交互式的 Shell（即 root Shell）。以下为示例：

```
$ docker ps
CONTAINER ID       NAMES
55ffedc5cc37       as150-router-10.150.0.254   ...
06d1693ad45f       as151-host-10.151.0.71      ...
debedefdfe18       as151-router-10.151.0.254   ...
265c098a6e19       as150-host-10.150.0.71      ...

$ docker exec -it 265 /bin/bash
#
```

方便起见 “docker ps”命令的输出很长，其中大部分输出信息对我们来说并不太有用。我们可以使用 --format 选项来限制打印字段。这会使得该命令相当长。由于我们将会频繁地使用这个命令，因此我们在 SEED VM 的 .bashrc 文件中创建了以下别名。更进一步地，我们也将会频繁地运行命令“docker exec”，所以我们也希望为这个命令创建别名。因为它需要一个参数，所以我们在 .bashrc 文件中添加了一个 Shell 函数定义。

```
alias dockps='docker ps --format "{{.ID}}  {{.Names}}" | sort -k 2'
docksh() { docker exec -it $1 /bin/bash; }
```

"sort -k 2" 命令将根据第二列将输出排序。有了上述别名和函数，我们只需执行以下命令即可列出所有正在运行的容器。

```
$ dockps
265c098a6e19  as150-host-10.150.0.71
55ffedc5cc37  as150-router-10.150.0.254
06d1693ad45f  as151-host-10.151.0.71
debedefdfe18  as151-router-10.151.0.254
```

有了这些 ID，我们就可以使用 docksh 函数在选定的容器上运行 bash 命令。我们无需输入完整的 ID 字符串。只要它们是唯一的，那我们只需输入首几位字符即可。

```
$ docksh 26
root@265c098a6e19:/#
```

测试（您的任务） 请进入 AS150 的主机。尝试 ping 同一自治系统内的路由器，并也 ping AS151 主机，记录观察结果（此处需要截图）。

```
# ping 10.150.0.254
# ping 10.151.0.71
```

4 任务 2. 互联网交换点与互联

从上一个任务中，我们可以看到尽管容器在同一自治系统内可以互相访问，但它们不能到达其他自治系统的机器。这是因为我们还没有连接这些两个自治系统。在这个实验中，我们将连接它们，使得彼此能够相互访问。

4.1 任务 2.1 连接电缆

两个自治系统可以通过直接或间接（即通过另一个自治系统）的方式互联。在本任务中，我们将会将两个自治系统直接连接起来。这意味着它们的网络需要在物理意义上被连接。这被称为 *对等互连*。对等互连可以在数据中心（在此两个自治系统相互连接）秘密地完成，也可以在一个公共的地点（该地点为自治系统提供对等互连的设施）进行对等互连。这种公开的场所称为互联网交换点（IXP）或互联网交换中心（IX）。

实际中的互联网交换中心可能会相当复杂，但其核心就是一个交换机（一个 LAN）。想要与这其中的其他自治系统进行对等互连的自治系统会直接将它们的 BGP 路由器连接到这个 LAN。在这个任务中，我们将在我们的仿真器中设置互联网交换中心，并在其中让 `AS150` 与 `AS151` 进行互联。

为了实现这个功能，`AS150` 和 `AS151` 都需要在互联网交换中心内放置一个 BGP 路由器。它们将会被连接到互联网交换中心提供的 LAN 上。对于这个互联网交换中心，网络是 `10.100.0.0/24`。这些 BGP 路由器也会与自己的网络相连，因此每个路由器都有两个网络接口和两个 IP 地址。他们都在 Compose 文件当中被指定。为了方便起见，我们列举了它们的 IP 地址如下：

```
Router/Host      Interface 1         Interface 2
-------------------------------------------------
AS150 Router     10.150.0.254        10.100.0.150
AS151 Router     10.151.0.254        10.100.0.151
```

每个 BGP 路由器的 BGP 配置文件都被放置在 `ix100` 文件夹中。它们将在构建 BGP 路由器容器镜像时使用。

任务（您的任务） 运行这个仿真器，从 `AS150` 主机 `ping` `AS151` 的主机，报告你的观察结果，并提供截图。使用 `mtr` 命令进行路线追踪，查看你的数据包去向。

```
# mtr -n 10.151.0.71
```

4.2 任务 2.2 直接互联

你会发现即使在硬件层面，`AS150` 和 `AS151` 的主机仍然无法联系到彼此，它们的网络是通过 `IX100` 互联网交换中心连接起来的，这两个自治系统内部的路由器仍然不知道其他自治系统内部的网络，或是通过其他自治系统能够到达的网络。

这是因为我们尚未实现互联网交换的软件部分，即 BGP。BGP 被认为是互联网的 “粘合剂”。在这个任务中，我们将设置 BGP，使得 `AS150` 和 `AS151` 之间可以交换有关它们网络和可以到达的网络的相关信息。这些信息将会被路由器用于传递数据包。

我们为 BGP 使用的软件叫做 BIRD。它已经在每个路由器容器中启动了。BIRD 从 `/etc/bird/` 文件夹中读取 `bird.conf` 配置文件。以下为 `AS150` 的 BGP 路由器的配置文件内容:

Listing 1: `AS150` 的 `bird.conf` 文件

```
protocol device {
}

protocol direct {
    interface "eth0";
}

protocol kernel {
    import none;
    export all;
}
```

为了在 BIRD 中定义路由协议，我们使用的是 `protocol` 关键词。BIRD 支持多种不同的路由协议，如 BGP 和 OSPF。一个 `protocol` 可以将导入或导出路由到 BIRD 的路由表中。对于这个实验来说，我们将会使用 `all` 或 `none`。在未来的 BGP 实验中，我们将在此添加过滤器，以便你设置规则来决定哪些路由可以被导入或导出。

- `device protocol` 并不是一个真正的路由协议。它不生成任何路由也不接受任何路由。它仅用于从内核获取网络接口信息。没有这个 `device protocol`，BIRD 将不会知道任何网络接口的信息。它甚至无法运行 BGP，因为它不知道如何得到与其进行 BGP 对等方的 IP 地址。因此，此部分是必需的。
- `direct protocol` 用于根据一组接口生成直接连接网络的路由。在 `direct protocol` 块中，*interface* 关键词用于从一个接口生成路由。在这种情况下，`AS150` 的 BGP 路由器使用 `eth0` 连接到 `AS150` 的内部网络（`10.150.0.0/24`）。因此，这个 `direct protocol` 块将为 `10.150.0.0/24` 生成路由条目。BGP 将向对等方传送此网络前缀。
- `kernel protocol` 并不是像 BGP 或 OSPF 这样的真正的路由协议。它将 BIRD 的路由表与内核的路由表连接起来，而不是与网络中的其他路由器通信。实际使用的路由是内核的路由表，而不是 BIRD 的路由表。这个协议用于通过从 BGP 对等方接收到的路由来设置内核的路由表。这里 `import none` 表示 BIRD 的路由表不会从内核的路由表中导入任何内容；`export all` 表示 BIRD 的路由表将向内核的路由表导出所有路由。这就是 BGP 路由器如何利用 BGP 协议收集的数据来设置其路由表的原理。

向 `bird.conf` 添加 BGP 协议 目前为止，我们已经展示了如何生成路由以及如何将 BIRD 的路由表用于设置内核路由表。但我们还没有告诉 BIRD 要运行 BGP 协议。我们在 `bird.conf` 中添加以下条目以在每个 BGP 路由器上运行 BGP 协议:

```
protocol bgp {
    import all;
```

```
    export all;

    local    10.100.0.150 as 150;
    neighbor 10.100.0.151 as 151;    ← BGP 对等方
}
```

BGP 协议中的 local 选项设置了本地 IP 地址和 BGP 会话的自治系统号。语法结构为 "local [ip_address] as <asn>", [ip_address] 部分是可选的, 但它可以使配置看起来更清晰, 并防止 BGP 路由在它有多个 IP 地址时选择错误的 IP 地址用于 BGP 会话。这个 IP 地址应当为互联网交换中心网络的 IP 地址。

BGP 协议中的 neighbor 选项设置邻居的 IP 地址和 BGP 会话的自治系统号。这才是实际建立对等关系的部分。在这个例子当中, 我们建立了 AS150 和 AS151 之间的 BGP 会话, 以便它们可以使用 BGP 协议交换路由信息。相似地, 在此情况下 IP 地址应该选择互联网交换中心网络的 IP 地址。

测试 (您的任务) 请在 IX100 内部的 AS150 和 AS151 的 BGP 路由器中添加 "protocol bgp" 块, 使得这两个路由器之间可以建立 BGP 会话并开始交换路由信息。启动所有容器, 测试两个自治系统内的主机是否能够互相访问。如果您已经正确完成了此任务, 则它们应该能相互访问。请提供截图和解释。

4.3 任务 2.3 通过路由服务器对等互联

在公共的互联网交换点中, 自治系统希望与其他多个自治系统进行对等互连。假设我们有 N 个自治系统, 并且它们想彼此进行互连。如果使用上一个任务中的方法, 每个对等的自治系统需要建立一个对等互连关系。这将会相当复杂。

大多数互联网交换点提供了一种简化这种方法的机制。它们提供一种特殊的服务器称为 路由服务器。所有这些 N 个自治系统只需与该路由服务器进行互联即可。当路由服务器通过 BGP 收到某参与者的路由时, 它会将该路由重新分发给其他连接的参与者。路由服务器的功能类似于多播技术: 任何发送至路由服务器的 BGP 路由都会被其所有与其进行对等互连的路由器接收。

需要注意的是, 路由服务器不是一个真正的 BGP 对等方, 并且其行为与真实的 BGP 对等方不同。最重要的一点是, 它不会在其路径中添加自己的自治系统编号, 也不会更改路径中的 nexthop 属性。对于其他 BGP 路由器来说它是透明的, 不会影响 BGP 协议的结果。通过路由服务器进行对等互连与直接进行对等互连是等效的。其主要目的单单只是使较多 BGP 路由器之间的对等互连更加简单。

在这个任务中, 我们将修改 AS150 与 AS151 之间的对等互连, 因此它们可以通过 IX100 提供的路由服务器进行对等互连。我们只需要做出一行改动。该路由服务在 IX100 的 IP 地址为 10.100.0.100。

```
neighbor 10.100.0.100 as 100;    ← 与路由服务器进行对等互连
```

路由器的 BIRD 配置 路由服务器需要在其 BIRD 配置文件中指定其对等方。它应该为每个对等方都有一个 "protocol bgp" 条目。在此条目中, 我们添加了 "rs client" 选项, 告诉 BIRD 不要在自治系统路径前预置互联网交换中心自己的自治系统号, 也不更改自治系统路径的 nexthop 属性。这样, 路由服务器的信息不会被添加到和其对等方交换的自治系统路径中。

所有这些 "protocol bgp" 条目将具有相同的内容，只是 neighbor 选项不同。在 BIRD 配置中，我们可以通过模板简化该配置。我们创建了一个名为 rs_peer 的 BGP 模板，并以此为基础构建所有 "protocol bgp" 条目。

```
template bgp rs_peer {
    import all;
    export all;
    rs client;
    local 10.100.0.100 as 100;
}

protocol bgp from rs_peer {
    neighbor 10.100.0.150 as 150;    ← 与 AS150 进行对等互连
}

protocol bgp from rs_peer {
    neighbor 10.100.0.151 as 151;    ← 与 AS151 进行对等互连
}
```

测试（您的任务） 请修改 IX100 中 AS150 和 AS151 的 BGP 路由器的 “protocol bgp” 块，并在路由服务器中添加相应的 “protocol bgp” 块。然后启动所有容器，从 AS150 主机到 AS151 主机进行路径追踪，并解释您的观察结果。具体来说，请将该结果与上一个任务（即不使用路由服务器直接进行将两个自治系统进行对等互连的情况）中的结果比较。

5 任务 3. 转送自治系统

到目前为止，如果两个自治系统之间需要连接，它们会在互联网交换点处进行对等互连。问题是两个不同地理位置的自治系统如何连接彼此。它们很难找到一个共同的位置来进行对等互连。为解决这个问题，我们需要一种特殊类型的自治系统。

这种自治系统在许多互联网交换点都有 BGP 路由器，这些路由器与其他自治系统之间进行对等互连。一旦数据包进入其网络，它将被从一个互联网交换点推送到另一个互联网交换点（通常通过一些内部路由器），最终交由另一个自治系统处理。这种类型的自治系统为其他自治系统提供转送服务。这就是即使这些自治系统之间不互相进行互连，但它们的主机仍可以相互通信的原因。这种特殊的自治系统被称为*转送自治系统*。在这个任务中，我们将向我们的互联网仿真器添加一个转送自治系统。

5.1 任务 3.1 在多个互联网交换点进行对等互连

下面是拓扑图。AS2 是一个转送自治系统，它在 IX100 处与 AS150 对等互连，在 IX101 处与 AS152 对等互连。其目的是为这两个自治系统提供转送服务，使得它们可以通过 AS2 联系到彼此。AS150 和 AS151 的对等互连关系保持不变，这些对等互连关系在图 1 中有所描述。所有本任务所需的文件都可以从 Task3_Transit 文件夹中找到。

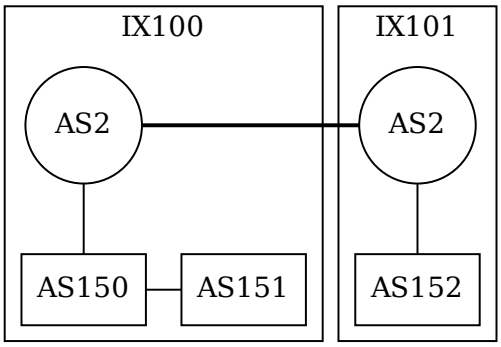

图 1: 任务 3 的互联网仿真器

我们需要为 AS2 设置两个 BGP 路由器。出于简化考虑，我们假设这两台路由器在同一 LAN 上 (10.2.0.0/24)。在现实世界中，这种情况通常不会发生。网络和 IP 地址分配如下（AS150 和 AS151 的信息保持不变)。

```
Router/Host              Interface 1     Interface 2
-------------------------------------------------------
AS2's Router @IX100      10.2.0.254      10.100.0.2
AS2's Router @IX101      10.2.0.253      10.101.0.2

AS152's Router @Ix101    10.152.0.254    10.101.0.152
AS152's Host             10.152.0.71         --
```

测试 启动所有容器。尝试从 AS150 的主机 ping 152 的主机，并且运行这两台主机的路径追踪。请展示你的观察结果。

在 AS2 的 BGP 路由器上运行 "ip route" 命令，看看是否能看到所有自治系统的条目（AS2 在我们的拓扑图中连接了所有自治系统)。以下是在 IX100 中 AS2 的路由器上的一个样例结果。你同样需要展示在 AS2 BGP 路由器上的结果。

```
# ip route
10.2.0.0/24 dev eth0 proto kernel scope link src 10.2.0.254
10.100.0.0/24 dev eth1 proto kernel scope link src 10.100.0.2
10.150.0.0/24 via 10.100.0.150 dev eth1 proto bird
10.151.0.0/24 via 10.100.0.151 dev eth1 proto bird
```

从上述结果可以看到，尽管 AS2 与 AS152 建立了对等互连关系，但 IX100 中 AS2 的 BGP 路由器并不知道如何发送信息到 AS152 的网络 10.152.0.0/24，但是该对等关系是在不同的互联网交换点中进行的。尽管 AS2 已经布线连接两个互联网交换点的 BGP，但他们之间并没有交换信息。我们缺少一个重要步骤。

5.2 任务 3.2 添加内部 BGP 对等互连

就像不同自治系统的 BGP 路由器进行对等互连一样，同一自治系统内的 BGP 路由器也需要交换信息，它们也需要互相进行对等互连并运行 BGP 协议来交换信息。在相同自治系统中的 BGP 路由器之间执行的 BGP 协议被称为 IBGP（内部 BGP）。

当我们用同一个自治系统号的两台路由器建立 BGP 会话时，它将被视为一个 IBGP 会话。当会话是在两个具有不同自治系统号的路由器之间进行时，此会话被视为一个 EBGP 会话（外部 BGP）。因此定义一个 IBGP 会话的方式与定义一个 EBGP 会话相同。

让我们在 IX100 和 IX101 中与 AS2 的 BGP 路由器中添加以下对等互连关系。在我们的设置当中，IX100 中的路由器的 IP 地址为 10.2.0.254，IX101 中的路由器的 IP 地址为 10.2.0.253。以下示例是 IX100 中的路由器条目。你需要对其他路由器进行一些更改。在示例中，我们给这个 BGP 会话命名为 ibgp，但你可以使用任何名称。

```
protocol bgp ibgp {
    import all;
    export all;

    local    10.2.0.254 as 2;
    neighbor 10.2.0.253 as 2;
}
```

注意事项 需要注意的是，IBGP 会话和 EBGP 会话不同，它具有几种不同的行为：

- 在 IBGP 会话中，当发送路由到对等方时，路由器不会在其 AS_PATH 中预置自己的自治系统号，并且也不会更改 nexthop 属性。**在您的实验报告中，请解释这是为什么**。
- BGP 路由器将不会从一个 IBGP 对等方收集的信息转发给另一个对等方；否则，会形成循环。这是因为 IBGP 不会在其自治系统路径中添加自己的信息，BGP 路由器无法判断它们的对等方是否已知道该自治系统路径。如果开启转发功能，BGP 路由器将持续向其对等方转发信息，从而形成循环。因为没有转发功能，通常在一个自治系统内部，所有 IBGP 对等方都会互相对等互联。

测试（您的任务） 在完成必要的更改后，请运行互联网仿真器。在 AS2 的 BGP 路由器上运行 "ip route" 命令，看看是否能看到所有自治系统的路由条目（在我们的拓扑图中，AS2 连接到了所有自治系统）。以下是 IX100 中 AS2 的路由器的一个样例结果。您需要在两个 AS2 的 BGP 路由器上展示自己的结果。

```
# ip route
10.2.0.0/24 dev eth0 proto kernel scope link src 10.2.0.254
10.100.0.0/24 dev eth1 proto kernel scope link src 10.100.0.2
10.150.0.0/24 via 10.100.0.150 dev eth1 proto bird
10.151.0.0/24 via 10.100.0.151 dev eth1 proto bird
unreachable 10.152.0.0/24 proto bird
```

这次我们确实看到了一条通往 AS152 网络的条目。这是与先前实验相比的进步，但该条目说“不可达”。让我们来找出原因。由于 IBGP 会话，现在 IX100 中 AS2 的 BGP 路由器已接收到有关 AS152 网络的信息。这就是为什么我们在内核路由表中看到该条目。

在任务 3.5 中，您将学习一个非常有用的工具 birdc。如果您运行此工具查看从 IBGP 会话接收到的路由信息，将会得到以下结果：

```
# birdc
bird> show route all
...
10.152.0.0/24   unreachable [ibgp 18:52:19 from 10.2.0.253] * (100/-) [AS152i]
   Type: BGP unicast univ
   BGP.origin: IGP
   BGP.as_path: 152
   BGP.next_hop: 10.101.0.152
   BGP.local_pref: 100
```

请注意 next_hop 属性。这个信息来自不同互联网交换点中的 BGP 路由器。正如我们在上面提到的，在 IBGP 会话中，路由器不会更改 next_hop 属性（在 EBGP 会话中，此属性会被更新，因为下一个跳转的地点发生了变化）。这个属性仍然指向位于 IX101 网络中的路由器 (10.101.0.152)，但 IX100 中的 BGP 路由器并未连接到该网络；因此它不知道如何到达 10.101.0.0 网络。这就是为什么它显示“不可达”的原因。它需要找到能够帮助将数据包转发至目标网络的路由器。

5.3 任务 3.4 添加内部路由

自治系统内的路由器之间需要互相通信，这样它们才能告诉彼此自己连接的网络是什么，从而其他自治系统能够确定到达这些网络的最佳路径。这就是在上一个任务中缺少的部分。

这种类型的路由协议称为 IGP（内部网关协议）。IGP 协议有很多种，包括 OSPF（一个连接状态路由协议）和 RIP（一个距离矢量路由协议）。这两种协议 BIRD 都支持。在这个任务中，我们将仅使用 OSPF 协议。

让我们向我们的 BIRD 配置文件中添加 OSPF 协议。在下面的配置中，我们创建了一个 "protocol ospf" 块并指定了一个表 t_ospf。这意味着所有 OSPF 路由信息都将存储在这个表中。在 "protocol bgp ibgp" 块中，我们告诉 BIRD 使用 t_ospf 表来解决从内部 BGP 对等方获得的路由的下一个跳转地点。

```
table t_ospf;  # 定义一个新的表

protocol bgp ibgp {
    import all;
    export all;

    igp table t_ospf;

    local    10.2.0.254 as 2;
    neighbor 10.2.0.253 as 2;
```

```
}

protocol ospf {
    table t_ospf;  # 如果没有这个条目, "master" 表将会被使用。

    import all;
    export all;

    area 0.0.0.0 {
        interface "eth0" {};
        interface "eth1" { stub; };
    };
}
```

OSPF 的详细配置可能会相当复杂，并且超出了这个实验的范围。编号为 ID 0 的区域称为骨干区域。所有其他区域都需要连接到此骨干区域。在我们的配置中，我们将所有的路由都放在骨干区域以简化设置。这一特别的 BGP 路由器 (IX100 中 AS2 的路由器) 连接到两个网络：一个是通过 eth0 ，另一个是通过 eth1。

来自这两个网络的信息将包含在 OSPF 协议中，这样其他方就知道如何到达这两个网络。然而 eth0 连接的是内部网络，而 eth1 是路由器用于连接外部网络（互联网交换点的网络）的接口。该网络由互联网交换点提供，用于对等互连。我们需要将此网络包含在 OSPF 协议中，这样内部路由器就知道如何到达此网络。但是我们将不会在此网络上运行 OSPF 协议，因为它们不属于 AS2，它们是外部的。您不希望向外界泄露内部网络信息，也不希望外部通过 OSPF 来操控你的内部路由。因此，我们不应该在这个外部网络上的路由器运行 OSPF 协议。我们只能在内部路由器运行 OSPF。这就是为什么我们使用 stub 的原因，意味着此网络的信息将用于 OSPF 协议中，但我们将不会在此网络上运行 OSPF。

测试（您的任务） 修改 BIRD 配置文件中 AS2 的所有 BGP 路由器，并然后运行仿真器。你可能需要等待一下，因为 OSPF 运行会占用一些时间。检查 AS2 的 BGP 路由器上的路由表，你应该能够看到来自 IX100 的机器可以到达 IX101 中的网络。请进行你的实验，提供结果并解释观察结果。

```
root@26aab6aab5d5:/# ip route
default via 10.2.0.1 dev eth0
10.2.0.0/24 dev eth0 proto kernel scope link src 10.2.0.254
10.100.0.0/24 dev eth1 proto kernel scope link src 10.100.0.2
10.150.0.0/24 via 10.100.0.150 dev eth1 proto bird
10.151.0.0/24 via 10.100.0.151 dev eth1 proto bird
10.152.0.0/24 via 10.2.0.253 dev eth0 proto bird
```

5.4 任务 3.5 查看 BGP

在这个任务中，我们将使用工具来进一步研究 BGP 协议。BIRD 提供了一个命令行工具 birdc，可以用于与 BIRD 交互来检查路由和会话。它相当容易使用。首先在路由器上获得一个 Shell，并输入

birdc。你将进入 BIRD 的交互式 Shell。实验网站上提供了一篇关于 birdc 使用手册的链接。

在 BIRD 的交互式 shell 中，可以在任何时候按 ? 键获取与当前上下文相关的帮助。例如，如果你键入 s ?，它会显示所有以字母 s 开头的命令。如果没有以这个字母开头的命令，则会显示所有命令。在本任务中，我们将仅使用 show, disable 和 enable 命令。

show 命令允许查看协议和路由的详细信息。enable 和 disable 命令可以用于启用/禁用协议。在以下示例中，我们首先列出了所有正在运行的协议，然后禁用了 BGP 会话 bgp1。你可以看到禁用前后的路由表变化。

```
bird> show protocol
name     proto    table    state  since       info
kernel1  Kernel   master   up     14:22:49
device1  Device   master   up     14:22:49
direct1  Direct   master   up     14:22:49
bgp1     BGP      master   up     14:22:53    Established

bird> show route
10.2.0.0/24        via 10.101.0.2 on eth1 [bgp1 15:30:16] * (100) [AS2i]
10.150.0.0/24      via 10.101.0.2 on eth1 [bgp1 15:30:16] * (100) [AS150i]
10.151.0.0/24      via 10.101.0.2 on eth1 [bgp1 15:30:16] * (100) [AS151i]
10.152.0.0/24      dev eth0 [direct1 14:22:50] * (240)

bird> disable bgp1
bgp1: disabled
bird> show route
10.152.0.0/24      dev eth0 [direct1 14:22:50] * (240)
```

show 命令中有几个有用的选项。我们在下面总结了一些。更多详细信息请参阅实验网站上的手册。

- show route all: 打印路由的详细信息。
- show protocol all bgp1: 打印 BGP 会话（bgp1）的详细信息。

嗅探 BGP 数据包 为了观察 BGP 协议的工作方式，我们可以使用 Wireshark 捕获所有协议数据包。在我们的仿真器中我们创建了多个网络。实际上，我们的主机 VM 连接到了所有这些网络，这使得我们的生活变得更加容易。分配给 VM 的 IP 地址是 .1。举个例子来说，在仿真器中创建的 10.2.0.0/24 网络，VM 的 IP 地址为 10.2.0.1。因此，如果我们想要捕获某个特定网络的数据包，只需选择相应的接口即可。

当我们在 Wireshark 中选择接口时，将会发现接口名称并不能告诉我们它属于哪个网络。我们可以运行 "docker network ls" 命令找到映射关系，然后使用该映射在 Wireshark 中选择正确的接口。

```
$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
...
537a85e6dfb8        seed_as2_net        bridge              local
16ba349d78a7        seed_as150_net      bridge              local
```

```
4f4061c87fcd        seed_as151_net      bridge              local
6ccf69ad6c00        seed_as152_net      bridge              local
...
```

您的任务 使用 `birdc` 和 Wireshark 完成以下任务。

- 在 `AS152` 路由器上运行 “show route” 命令，并解释您看到的内容。
- 禁用一个 BGP 会话，然后启用它。使用 Wireshark 捕获所有 BGP 数据包，仔细查看这些数据包并汇报您的发现。

6 任务 4. 扩展我们的互联网仿真器

我们现在已经学习了如何构建一个互联网仿真器。让我们通过添加更多组件来扩展这个仿真器。我们将添加一个互联网交换中心 (`IX102`)，一个新的转送自治系统 (`AS3`)，以及两个残桩自治系统 (`AS160` 和 `AS161`)。它们的对等互连关系如图 2 所示。

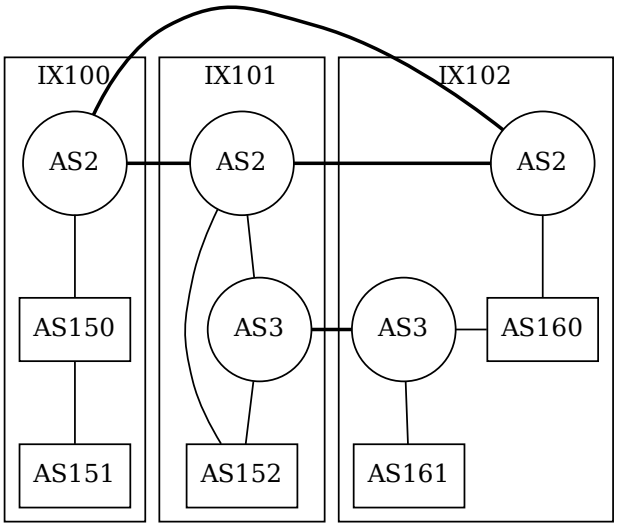

图 2: 任务 4 的互联网仿真器

提供给过渡系统的 BGP 路由器（AS2 和 AS3）的 IP 地址如下。这些是根据我们约定分配的。

```
Router/Host             Interface 1     Interface 2
-------------------------------------------------------
AS2's Router @IX100     10.2.0.254      10.100.0.2
AS2's Router @IX101     10.2.0.253      10.101.0.2
AS2's Router @IX102     10.2.0.252      10.102.0.2
-------------------------------------------------------
AS3's Router @IX101     10.3.0.254      10.101.0.3
AS3's Router @IX102     10.3.0.253      10.102.0.3
```

对于其他自治系统的网络，学生应该选择一些实际的网络（例如某个真实公司或大学的网络）。例如，`128.230.0.0/16` 属于纽约雪城大学，我可以将其用于 `AS150`。需要注意的是，学生应该独立选择这些网络。两个学生选取完全相同一组的 5 个网络的概率非常低。如果发生这种情况，教师应注意检查实验报告以防止抄袭。此外，为了使学生难以使用上几个学期的成果，教师应为其中一个自治系统随机选择网络 ID，并在每学期中改变这一选择。学生可以向教授询问该 ID。

请构建此仿真器、运行它并提供测试结果来展示您的互联网仿真器是否按预期工作。在实验报告中，请展示您添加到 BIRD、Docker 和 Docker Compose 的所有配置文件或内容。

7 提交

你需要提交一份带有截图的详细实验报告来描述你所做的工作和你观察到的现象。你还需要对一些有趣或令人惊讶的观察结果进行解释。请同时列出重要的代码段并附上解释。只是简单地附上代码不加以解释不会获得学分。