

看懂 lab11.py 前需要的知识：

1. 局域网（LAN）

局域网（LAN, Local Area Network）是指在一个较小的地理范围内（如一个家庭、办公室或学校）通过网络设备连接起来的计算机和其他设备的集合。局域网允许设备之间进行通信和共享资源（如文件、打印机、互联网等）。

局域网的特点：

- **范围小：**局域网的覆盖范围通常限于一个建筑物、一个楼层或一间办公室等。它可以连接几十台或几百台设备。
- **高速度、低延迟：**局域网通常提供较高的传输速度和较低的延迟，适合设备之间的高速数据交换。

局域网的作用：

- 在局域网内，设备可以轻松共享文件、打印机、硬盘等资源。
- 局域网可以连接到 **广域网（WAN）** 或 **互联网**，让局域网内的设备可以访问外部网络。

举例：

比如，你家里的路由器连接了几台电脑、手机和智能电视，这些设备都在同一个局域网内。你可以在电脑和手机之间共享文件，或者将手机上的视频传输到电视上观看。

2. MAC 地址

MAC 地址（Media Access Control Address）是网络设备的唯一标识符。每一台网络设备（如电脑、手机、交换机、路由器、打印机等）都有一个全球唯一的 MAC 地址。

MAC 地址的特点：

- **硬件地址：**MAC 地址是由设备的网络适配器（如网卡、Wi-Fi 模块等）在制造时被硬编码进设备的硬件中，它在设备的一生中通常不会改变。
- **由 6 个字节（48 位）组成，**通常以十六进制格式表示，形式如下：**00:14:22:01:23:45**。前 3 个字节通常表示设备的制造商（称为 **OUI**），后 3 个字节是设备的序列号，确保每个设备的 MAC 地址是唯一的。
- **局域网通信：**MAC 地址通常用于局域网中的通信，它帮助设备识别和转发数据包。

3. OSI 模型概述

在解释网络层之前，我们先了解一下 **OSI 模型**。OSI 模型分为七个层次，**从下到上**分别是：

1. **物理层（Layer 1）：**处理数据的物理传输（如电缆、光纤、无线信号等）。
2. **数据链路层（Layer 2）：**负责数据在同一网络中的传输和错误检测（例如 **MAC 地址**）。
3. **网络层（Layer 3）：**负责数据包的路由和转发（例如 **IP 地址、路由器**等）。
4. **传输层（Layer 4）：**确保数据可靠的传输（例如 **TCP、UDP 协议**）。
5. **会话层（Layer 5）：**确保会话维持。
6. **表示层（Layer 6）：**负责数据的编码和转换，确保不同系统之间的数据能够理解。

7. 应用层 (Layer 7)：直接为用户提供服务（例如 HTTP、FTP、SMTP 协议）。

3.1 “自下而上”是什么意思？

在 OSI 模型中，数据从发送方到接收方的传输是 **自下而上的**，也就是说，数据会从模型的底层逐层向上处理，直到达到目标应用。反之，从接收方传送过来的数据也会从应用层开始处理，逐层向下传递。

自下而上的流程：

1. 发送数据：

- 数据从应用程序开始，应用层把数据交给表示层（负责编码、压缩、加密等），然后表示层把数据交给会话层（管理会话），接着会话层把数据交给传输层（确保可靠性），然后传输层将数据传递到网络层（路由和传输），再到数据链路层（局域网内的传输），最后通过物理层传送到目标计算机。

2. 接收数据：

- 接收方的物理层接收到数据后，数据会逐层向上传递。数据链路层会去除数据帧头，网络层去除 IP 地址，传输层确保数据无误并交给会话层，接着会话层和表示层处理好数据后，最后传递给应用层，供用户使用。

举个简单的例子：

- 当你发送一个电子邮件时：
 - **应用层**：你使用邮箱客户端输入邮件内容。
 - **表示层**：邮件内容被编码成适合传输的格式。
 - **会话层**：确保邮件发送过程中的会话能够维持。
 - **传输层**：确保数据的可靠传输。
 - **网络层**：选择最佳路径，将数据发送到邮件服务器。
 - **数据链路层和物理层**：通过局域网或互联网传输邮件数据。
- 接收方收到邮件后，数据会按照相反的顺序（自上而下）被解码和处理，直到最终用户能够看到邮件内容。

3.2 应用层 (Layer 7)

应用层 (Application Layer) 是 OSI 模型的第七层，它直接与用户交互，提供各种网络服务和应用程序。应用层为用户提供各种网络服务，使用户能够访问网络资源。

应用层的作用：

- **提供服务**：应用层向用户提供服务，例如电子邮件、文件传输、网页浏览等。
- **协议实现**：应用层定义了应用程序之间如何通信的规则和协议，例如 HTTP（网页浏览）、FTP（文件传输）、SMTP（电子邮件）等。
- **用户交互**：应用层处理用户与网络的交互，确保用户输入的数据能够正确发送和接收。

3.3 表示层 (Layer 6)

表示层 (Presentation Layer) 是 OSI 模型的第六层，主要负责 **数据的编码、压缩和加密**，确保不同系统之间能够理解传输的数据。

表示层的作用：

- **数据编码**：表示层可以对数据进行编码，以便不同类型的计算机系统可以理解。例如，Windows 系统与 Linux 系统的文件编码方式可能不同，表示层负责处理这些编码差异。
- **数据压缩**：表示层可以对数据进行压缩，以减少数据的传输量，提高传输效率。
- **数据加密**：表示层也可以对数据进行加密，确保数据在传输过程中保持隐私和安全性。比如 HTTPS 就涉及加密。
- **数据解码**：当数据到达接收端时，表示层负责解码，以便应用程序能够正确地理解和处理数据。

举例：

- **图片格式转换**：如果你从 Windows 传输一个 JPEG 格式的图片到 Mac 系统，表示层可以确保不同系统使用的图像格式能够正确转换。
- **加密**：在访问银行账户时，HTTPS 加密协议会在表示层加密你的数据，确保信息传输时不被窃取。

3.4 会话层 (Layer 5)

会话层 (Session Layer) 是 OSI 模型的第五层，主要负责 **管理和控制不同应用程序之间的通信会话**。会话层的作用可以理解为：它负责建立、维护和终止应用程序之间的连接（也称为“会话”）。

会话层的作用：

- **会话管理**：会话层允许应用程序之间建立一个会话，以便它们能够进行多次交互。会话层**确保每个应用程序知道彼此的状态**（比如“开始”、“进行中”、“结束”），并且保证会话的顺序。
- **同步和恢复**：它可以为数据传输提供同步机制，确保长时间运行的任务能够正常完成。比如，如果传输过程中发生故障，会话层可以重新启动或恢复传输。
- **双向通信**：会话层可以支持双向的通信模式，即双方可以同时发送和接收数据。

举例：

- **视频会议**：如果你在和别人进行视频会议，会话层确保你们之间的连接可以维持，并管理会话的开始、进行和结束
- **文件传输**：如果你使用 FTP 传输文件，会话层会确保文件传输的开始、进程和结束

3.5 传输层（Layer 4）

传输层和网络层的区别：

特性	网络层（Layer 3）	传输层（Layer 4）
作用	负责在不同网络之间路由数据包，使数据包从源设备传输到目标设备。	负责端到端的通信，确保数据从源设备的一个应用程序传输到目标设备的相应应用程序。
处理的单位	数据包（Packet）	数据段（Segment，在 TCP 中）或 数据报（Datagram，在 UDP 中）。
地址	使用 IP 地址 标识源设备和目标设备。	使用 端口号 来区分应用程序或服务。
核心协议	IP 协议（IPv4、IPv6）、ICMP、ARP 等。	TCP、UDP 等协议。
主要设备	路由器（Router）	端系统（如计算机、服务器等）
数据可靠性	只负责将数据包传输到目标网络，不保证数据包的顺序和完整性。	提供数据可靠性（通过 TCP），包括重传、确认和顺序保证；或者快速传输（通过 UDP）。
通信类型	网络设备之间的通信（跨网络、跨子网）。	应用程序之间的通信（端到端）

- 网络层解决“到哪里去”的问题：

- 网络层根据 IP 地址 确定数据包的目标网络和设备。它通过路由选择最佳路径，将数据包送到目标网络。

- 传输层解决“给谁用”的问题：

- 数据到达目标设备后，传输层通过 端口号 确保数据交给目标设备上正确的应用程序。例如，你的电脑上同时运行了浏览器和邮件客户端，传输层会确保网页数据送到浏览器，邮件数据送到邮件客户端。

端口号（Port Number）是传输层用来标识设备上具体应用程序或服务的数字。

在一台设备上，可能运行着多个程序（例如：浏览器、聊天软件、邮件客户端等）。IP 地址只能确定数据发送到哪台设备，但无法区分设备上的程序。端口号 用来区分同一设备上的不同应用程序或服务。

举个例子：

- 你的电脑同时运行了浏览器（打开网页）和聊天软件（发送消息）。
- 目标服务器发回数据后，传输层会根据端口号把网页内容发送到浏览器，把聊天消息发送到聊天软件。

端口号是一个 16 位数字，范围从 0 到 65535，分为以下几类：

1. 系统端口（0-1023）：

- 被常见服务或协议占用，例如：
 - HTTP（网页浏览）：80
 - HTTPS（安全网页）：443
 - FTP（文件传输）：21

- SMTP（发送邮件）：25

- 这些端口通常是固定的，操作系统会为这些服务分配这些端口。

2. 用户端口（1024-49151）：

- 分配给用户应用程序或服务。例如：在线游戏、文件共享软件可能使用这些端口。

3. 动态端口（49152-65535）：

- 临时分配给应用程序，用于客户端和服务器的短期通信。例如，当你访问一个网页时，浏览器会临时使用一个动态端口与服务器通信。

举个例子：HTTP 请求

假设你在浏览器中访问一个网站，数据传输的过程如下：

1. 客户端：

- 你的电脑分配了一个临时动态端口（例如：52345），用于与服务器通信。发送到服务器 80 端口号

2. 服务器：

- 服务器监听端口 **80**，接收来自客户端的 HTTP 请求，记录客户端端口号 如 52345
- 服务器处理请求后，将网页数据返回给客户端，目标端口为客户端的临时动态端口（例如：52345）。

3. 数据传输：

- 传输层根据端口号确保数据从客户端的浏览器传到服务器的 HTTP 服务，并且将服务器响应的数据正确地传回到客户端浏览器。

为什么端口号有两个：80 和 52345？

1. 固定端口号（80）：服务器服务的“入口”

- 服务器使用固定端口号（比如 HTTP 的端口是 **80**，HTTPS 的端口是 **443**）来监听客户端的请求。
- 客户端访问服务器时，需要知道服务器的固定端口号，才能找到服务器上运行的具体服务。例如：
 - 80 端口用于 HTTP（网页）。
 - 25 端口用于 SMTP（邮件）。
- 服务器需要监听这些固定端口，才能随时处理来自客户端的请求。

2. 动态端口号（52345）：客户端的“出口”

- 客户端需要一个“临时端口”来与服务器通信，这个端口由客户端操作系统分配，称为**动态端口**。
- 动态端口是客户端用来标识这次通信的唯一标识符，服务器可以通过动态端口将返回的数据准确地传给客户端的特定程序

实例：发送消息到远程服务器

1. 网络层：

- 使用 **目标 IP 地址**（例如：192.168.1.10）找到目标服务器。
- 将数据包从源设备通过路由器和其他网络设备传输到目标设备。

2. 传输层：

- 使用 **目标端口号**（例如：80，表示 HTTP 服务）将数据传输到目标服务器上的正确服务。
- 使用 **源端口号**（例如：52345）确保目标设备知道如何将响应数据返回给源设备的正确程序。

传输层的核心功能：

1. 数据分段：

- 将应用层传下来的大块数据分成更小的“数据段”以便网络传输。

2. 可靠传输（由 TCP 提供）：

- 确保数据包能够正确到达目标，即使在网络中发生丢包、乱序等问题，也能保证数据的完整性。

3. 无连接传输（由 UDP 提供）：

- 不需要建立连接，直接发送数据，适合快速或实时的通信。

4. 端口号的使用：

- 传输层通过 **端口号** 来区分不同的应用程序。比如，浏览网页使用 **HTTP 协议（80 端口）**，发送邮件使用 **SMTP 协议（25 端口）**。

在传输层，最常用的协议是 **TCP（Transmission Control Protocol）** 和 **UDP（User Datagram Protocol）**。它们是网络中两种重要的传输方式。

(1) TCP 协议（可靠传输协议）

TCP 是一种面向连接的协议，它确保数据可靠地传输到目标设备。

TCP 的特点：

1. 三次握手（建立连接）：

- 在发送数据之前，TCP 通过 **三次握手** 过程建立一个可靠的连接。
- 这就像在打电话前说“你好”，确保双方都准备好了再开始交流。

2. 数据包顺序：

- TCP 确保数据包按正确的顺序到达。如果数据包乱序到达，TCP 会重新排序。

3. 丢包重传：

- 如果某些数据包在传输过程中丢失，TCP 会自动重传，确保数据完整。

4. 确认机制：

- 接收方在收到数据后，会发送一个确认消息（ACK）告诉发送方“我收到数据了”。

5. 流量控制：

- TCP 会根据网络状况动态调整发送数据的速度，避免网络拥塞。

TCP 的应用场景：

- 网页浏览 (HTTP/HTTPS)：浏览器通过 TCP 连接下载网页内容。
- 文件传输 (FTP)：传输文件需要确保数据完整，因此使用 TCP。
- 电子邮件 (SMTP)：邮件需要可靠传输，TCP 是首选。

(2) UDP 协议（快速传输协议）

UDP 是一种面向无连接的协议，它不会建立连接，而是直接发送数据。相比 TCP，它的传输速度更快，但不保证数据可靠。

UDP 的特点：

1. 无连接：
 - 发送数据前不需要建立连接，直接把数据包发出去。
2. 不保证数据可靠性：
 - UDP 不会确认数据是否成功到达，也不会重传丢失的数据。
3. 传输速度快：
 - 因为没有握手和确认机制，UDP 的传输效率比 TCP 更高。

UDP 的应用场景：

- 实时通信（视频会议、语音通话）：
 - 在实时场景中，即使丢失一部分数据（如语音或视频帧），用户也可以继续交流。
- 在线游戏：
 - 游戏场景需要快速传输数据，丢失一些位置数据不会影响整体游戏体验。
- DNS（域名系统）：
 - 查询域名对应的 IP 地址时，数据包很小且请求频繁，使用 UDP 更高效。

TCP 和 UDP 的对比

特性	TCP（传输控制协议）	UDP（用户数据报协议）
连接类型	面向连接，需要建立连接（例如三次握手）	面向无连接，不需要建立连接
可靠性	高：有确认、重传和排序机制，确保数据完整传输	低：不确认数据是否到达，不重传数据
速度	慢：传输可靠，但由于额外的控制机制速度较慢	快：无额外控制机制，速度快
数据顺序	有序：确保数据按正确顺序到达	无序：不保证数据顺序
应用场景	文件传输、电子邮件、网页浏览	视频会议、语音通话、实时游戏、DNS 查询

举个例子帮助理解

TCP 的例子：寄快递

想象你寄一个快递：

1. 你先打电话给对方确认（相当于三次握手）。
2. 快递员会把包裹送到对方家里，并要求对方签收（确认机制）。
3. 如果快递中途丢失，快递公司会重新发送包裹（重传机制）。
4. 所有包裹到达后，对方会把东西按正确顺序排列（数据顺序）。

UDP 的例子：写明信片

想象你寄一张明信片：

1. 你直接把明信片放进邮筒（无连接）。
2. 明信片送到对方时，没有人确认你是否收到了，也不会补发丢失的明信片（无确认、无重传）。
3. 明信片是单向发送的，没有多余的交流，但速度非常快。

3.6 网络层（Layer 3）的作用

网络层的主要作用是负责 **不同网络之间的数据传输**。它将来自上层的数据（通常是传输层的数据段）封装成 **数据包（Packet）**，并根据目标地址（通常是 **IP 地址**）来选择数据的最佳传输路径。

网络层中的关键设备是 **路由器（Router）**，它决定数据包如何从源设备传送到目标设备。

网络层的核心功能：

1. **路由（Routing）**：网络层决定了数据包应该如何从一个网络传输到另一个网络。它通过 **路由表** 来选择最佳的传输路径。路由器使用路由协议（如 **RIP、OSPF、BGP**）来交换和更新路由信息。
2. **IP 地址**：网络层使用 **IP 地址** 来标识和定位网络中的设备。每个设备都有一个唯一的 IP 地址，网络层通过这个地址将数据包送到正确的目标设备。
3. **分段与重组**：由于不同网络的最大传输单元（**MTU**）可能不同，网络层可能需要将数据包进行分段，直到它适应传输的网络，并在目标设备上重组。
4. **流量控制与拥塞管理**：虽然 **流量控制通常由传输层负责**，网络层也能采取一些策略来避免网络拥塞和数据丢失。

常见问题：网络层使用 **IP 地址** 来标识和定位网络中的设备。每个设备都有一个唯一的 IP 地址，网络层通过这个地址将数据包送到正确的目标设备。这个每个设备是路由器吗？

答：不一定是路由器。IP 地址可以标识 **任何连接到网络的设备**，不仅仅是路由器。IP 地址是用来标识网络中的每个设备，这些设备包括但不限于：

1. **计算机**（台式机、笔记本、服务器等）
2. **智能手机**（连接 Wi-Fi 或移动网络的设备）
3. **打印机**（网络打印机）
4. **物联网设备**（如智能家居设备、摄像头、智能电视等）

- 5. 交换机（支持管理功能的交换机，如 L3 交换机）
- 6. 路由器（作为网络的“中间人”，路由器也需要 IP 地址进行通信）

设备的 IP 地址通常是局域网内的私有 IP 地址（不包括路由器的公网 IP 地址），因为大多数设备都是通过局域网连接到网络的。

IP 地址

IP 地址（Internet Protocol Address）是用于标识网络中设备的地址，类似于你家中的门牌号。它在网络层（Layer 3）工作，是设备在互联网或广域网（WAN）中唯一的标识符。

IP 地址的特点：

- 分为 IPv4 和 IPv6 两种：
 - IPv4 地址是由 4 个数字组成，格式为 xxx.xxx.xxx.xxx，其中每个数字的范围是 0 到 255。例如：192.168.1.1。
 - IPv6 地址是由 8 个 16 进制数构成，格式为 xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx，提供了比 IPv4 更多的地址空间。
- 网络通信：IP 地址用于不同网络之间的通信，确保数据包能够从源设备发送到目标设备。网络中的每个设备都需要有一个唯一的 IP 地址，特别是在互联网上。

IP 地址的分类：

- 私有 IP 地址：通常用于局域网内的设备，这些地址在互联网中是不可路由的。例如：192.168.x.x、10.x.x.x。
- 公网 IP 地址：用于互联网上的设备，每个设备都需要一个唯一的公网 IP 地址。互联网服务提供商（ISP）会分配这些地址。

举例：

- 局域网中的 IP 地址：你家里的路由器通常会分配一个私有 IP 地址给每个设备，例如：192.168.1.10、192.168.1.11 等。设备通过这些私有 IP 地址互相通信。
- 互联网上的 IP 地址：当你在浏览器中输入网址时，实际上是通过 IP 地址来访问网站的。例如，www.google.com 的 IP 地址可能是 172.217.5.110。

MAC 地址与 IP 地址的区别：

特性	MAC 地址	IP 地址
作用	用于同一局域网内设备间的通信	用于标识不同网络或子网中的设备
层次	数据链路层（Layer 2）	网络层（Layer 3）
唯一性	唯一标识每个网络设备，全球唯一	在同一网络中唯一，互联网上是全球唯一的
是否可变	固定不变，设备出厂时由厂商设定	可变，设备可以动态获取或设置 IP 地址

特性	MAC 地址	IP 地址
格式	48 位，通常表示为 6 个十六进制数 (例如 00:14:22:01:23:45)	IPv4: 32 位，格式为 xxx.xxx.xxx.xxx ; IPv6: 128 位，格式为 xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx: xxxx

IP 地址和路由器的关系

虽然路由器也有 IP 地址，但它的角色和普通设备不同。

路由器通常有 **两个或更多的 IP 地址**，分别用于不同的网络。

路由器的 IP 地址：

1. 内网 IP 地址 (LAN IP 地址)：

- 路由器作为局域网的网关，**拥有一个内网 IP 地址**。例如：**192.168.1.1**。
- 局域网中的设备（电脑、手机等）通过这个内网 IP 地址将流量发送到路由器。

2. 公网 IP 地址 (WAN IP 地址)：

- 路由器连接到互联网时，会被分配一个公网 IP 地址（由互联网服务提供商 ISP 提供），**用于与外部网络通信**。
- **所有局域网设备访问互联网时，都会通过路由器的公网 IP 地址**。

数据包传输的过程

假设你在电脑上访问一个网站 (<http://example.com>)，以下是网络层如何处理 IP 地址的步骤：

1. 源设备（你的电脑）：

- 你的电脑有一个私有 IP 地址（如 **192.168.1.100**）。
- **网络层会把你的电脑的 IP 地址作为数据包的源 IP 地址**。

2. 目标设备（网站服务器）：

- 网站服务器有一个公网 IP 地址（如 **93.184.216.34**）。
- **网络层会把这个 IP 地址作为数据包的目标 IP 地址**。

3. 中间路由器：

- **数据包会先从你的电脑发送到局域网的路由器**。
- **路由器会查看目标 IP 地址（93.184.216.34），并决定如何转发数据包**。
- **如果目标在互联网上，路由器会通过其公网 IP 地址将数据包发送给上一级网络**。

4. 到达最终目标设备：

- 经过多级路由，数据包最终到达目标服务器，服务器会根据你的请求返回数据。

网络层常见协议

网络层主要使用 **IP 协议** 来传输数据。这里有几个常见的协议：

- **IP 协议 (Internet Protocol)**：用于在网络中标识和定位设备。IP 地址由 **IPv4** (32 位) 和 **IPv6** (128 位) 两种格式组成。
- **ICMP 协议 (Internet Control Message Protocol)**：用于在网络中发送控制消息，常见的用途有 **Ping** 命令，用来检测设备是否连通。
- **ARP 协议 (Address Resolution Protocol)**：用于将 **IP 地址** 映射到 **MAC 地址**，这是局域网中数据转发所需的信息。
- **路由协议**：用于动态地选择数据包的转发路径，例如 **RIP (Routing Information Protocol)**、**OSPF (Open Shortest Path First)**、**BGP (Border Gateway Protocol)** 等。

4. 交换机 (Switch)

交换机 是一种常见的网络设备，它主要用于 **局域网 (LAN)** 中的设备间连接。交换机的作用是将数据包从一个设备转发到另一个设备。

交换机的工作原理：

交换机工作时主要依赖于 **MAC 地址 (媒体访问控制地址)** 来转发数据包。每个网络设备（如电脑、打印机、手机等）都有一个唯一的 **MAC 地址**。交换机通过学习并记录各设备的 **MAC 地址** 和它们连接的 **端口**，从而决定如何转发数据包。

交换机的功能：

- **学习功能**：当交换机接收到一个数据包时，它会查看数据包的 **源 MAC 地址**，并将其记录在 **MAC 地址表** 中。这样，交换机就知道该数据包来自哪个设备，并且可以记住该设备连接在哪个端口上。
- **转发功能**：当交换机收到一个数据包时，它会根据数据包的 **目标 MAC 地址** 查找 **MAC 地址表**，找到对应的端口并将数据包转发到正确的设备。如果目标设备不在表中，交换机会将数据包广播到所有端口，直到找到目标设备为止。
- **避免冲突**：交换机通常有多个端口，能够同时处理多个数据流，避免了传统集线器 (Hub) 带来的数据冲突问题。

交换机举例：

想象一个办公室中有多个电脑、打印机等设备，它们都通过交换机相连。如果电脑 A 想给打印机发送数据，交换机就会根据数据包中的目标地址（打印机的 **MAC 地址**），将数据包准确地转发到打印机所在的端口。

5. 路由器 (Router)

路由器 是一种用于 不同网络之间 互联的设备。路由器的作用是 根据目的地址选择最佳路径 将数据包从一个网络转发到另一个网络。它主要工作在 网络层，而交换机主要工作在 数据链路层。

路由器的工作原理：

路由器通过查看数据包的 目标 IP 地址 来确定数据包应该被发送到哪个网络。它通过 路由表 来选择数据包的最佳路径。

路由器的功能：

- **路径选择：**路由器通过计算不同路径的代价（例如：带宽、延迟、网络负载等）来选择数据包的转发路径。它可以在多个网络之间找到最优的传输路线。
- **网络分段：**路由器可以将网络分成多个子网，并通过 子网掩码 确定数据包是否需要被转发到另一个子网。
- **NAT（网络地址转换）：**路由器还可以执行 NAT 功能，它将一个私有网络中的设备的 IP 地址转换为公共 IP 地址，从而实现设备之间的互联网连接。

路由器举例：

想象一下你在家有一台电脑和一台打印机，它们通过路由器连接到互联网。如果你要访问一个网站，路由器就会将数据包从你的本地网络转发到互联网，并确保数据包能够到达网站的服务器。同样地，路由器会根据不同的网络和地址来选择最佳路径，以确保信息可以顺利传递。

交换机与路由器的区别：

特性	交换机 (Switch)	路由器 (Router)
工作层次	数据链路层 (Layer 2)	网络层 (Layer 3)
作用	在同一网络内，负责连接设备并转发数据包	在不同网络之间转发数据包，选择最佳路径
转发依据	MAC 地址	IP 地址
连接的设备	通常用于局域网 (LAN) 内部设备之间的连接	用于连接不同网络、不同子网或网络和互联网之间的连接
例子	连接计算机、打印机、服务器等	将本地网络连接到互联网，或连接多个子网

6. 软件定义网络 SDN

软件定义网络（SDN, **Software-Defined Networking**）是一种新型的网络架构，它将传统网络中的**控制平面**和**数据平面**分离，从而实现网络的集中控制和灵活管理。SDN 的核心理念是通过软件来控制网络，而不是依赖硬件设备的固定逻辑。

6.1 SDN 的核心概念

(1) 传统网络 vs. SDN

- **传统网络：**
 - 每个网络设备（如交换机、路由器）既负责处理数据流量（数据平面），也负责决定如何转发数据包（控制平面）。
 - 配置复杂，灵活性差。管理员需要逐台设备手动配置规则，适应新需求耗时长。
- **SDN 网络：**
 - SDN 将网络设备的**数据平面**和**控制平面**分离。
 - **数据平面：**负责实际转发数据包（由交换机、路由器等硬件执行）。
 - **控制平面：**负责制定转发规则，由一个中央**控制器**通过软件集中管理所有设备。
 - 灵活高效。管理员只需在控制器上配置规则，所有网络设备会按照控制器的指令工作。

(2) SDN 的三层架构

1. 应用层（Application Layer）：

- 包括各种网络应用程序，如流量优化、网络监控、安全防护等。
- 网络管理员或开发者可以通过编程方式使用 SDN 提供的 API 开发这些应用。
- **作用：**用户与网络交互的接口，定义网络需要完成的任务。

2. 控制层（Control Layer）：

- 由**SDN 控制器**组成，是 SDN 的“大脑”。
- **控制器通过南向接口（如 OpenFlow 协议）管理数据平面的网络设备。**
- **作用：**
 - 集中控制网络设备，制定转发规则。
 - 收集设备的状态信息并动态调整网络策略。

3. 数据层（Data Layer）：

- 包括交换机、路由器等设备，负责实际处理和转发数据流量。
- **作用：**严格按照控制器下发的规则转发数据包，不自行做决策。

6.2 SDN 的特点

1. 集中控制：

- 控制器集中管理网络中的所有设备，管理员可以通过控制器查看和调整整个网络的行为。

2. 可编程性：

- 网络功能可以通过软件编程实现，管理员可以轻松添加或修改网络规则，而不需要更换硬件设备。

3. 灵活性：

- SDN 网络可以根据需求动态调整。例如，管理员可以设置流量优先级，或根据实时流量负载优化数据路径。

4. 网络虚拟化：

- SDN 支持将物理网络抽象为多个虚拟网络，每个虚拟网络可以为特定用户或应用提供定制服务。

5. 开放接口：

- SDN 使用标准化的接口（如 OpenFlow）与网络设备通信，使得不同厂商的设备可以协同工作。

6.3 SDN 的核心技术：OpenFlow

OpenFlow 是一种通信协议，用于控制器和网络设备之间的交互。它定义了控制器如何下发规则给数据平面的设备，以及如何从设备收集网络状态信息。

OpenFlow 的主要功能：

- 控制器可以通过 OpenFlow 协议向交换机安装、修改或删除流表规则。
- 交换机将不匹配规则的数据包发送给控制器处理（称为 Packet-In）。
- 控制器可以通过 Packet-Out 消息直接下发转发指令。

OpenFlow 中的基本概念

1. **数据包匹配 (Packet Matching)**：在 OpenFlow 中，交换机会根据预定义的“匹配规则”来决定如何处理数据包。每个规则通常会检查数据包的头部信息，如 MAC 地址、IP 地址、端口号等。
2. **流表 (Flow Table)**：交换机的流表中保存了一些规则，决定了如何处理网络中的数据包。每一条流表规则包含了：
 - **匹配字段**（例如：源 MAC 地址，目标 IP 地址等）
 - **动作**（例如：将数据包转发到哪个端口，或者丢弃数据包）
3. **流表操作**：
 - **流表添加 (Flow Mod)**：向交换机的流表中添加新规则。
 - **流表修改**：修改现有的流表规则。
 - **流表删除**：删除流表中的规则。

6.4 SDN 的优势

1. 网络管理更简单：

- 传统网络中需要逐台配置设备，而 SDN 中可以通过控制器统一配置。

2. 适应需求更灵活：

- SDN 可以快速响应网络流量的变化，比如调整带宽分配或重新配置路径。

3. 网络资源利用率更高：

- 控制器可以实时监控网络状态，并动态分配网络资源，避免过载或资源浪费。

4. 支持自动化和智能化：

- 结合 AI 和机器学习，SDN 可以实现自动化网络优化，例如动态检测和修复故障。

5. 支持网络创新：

- 网络管理员或开发者可以通过编程实现新的功能，无需更换硬件设备。

6.5 SDN 的实际应用场景

1. 数据中心网络：

- 数据中心的流量需求多变，SDN 可以根据实时流量动态分配网络资源，避免拥堵。

2. 广域网（WAN）优化：

- 企业可以通过 SDN 控制器优化跨分支机构之间的数据传输路径，节省带宽成本。

3. 安全与防护：

- SDN 可以实现集中式安全管理，例如流量监控、入侵检测、DDoS 防护等。

4. 网络虚拟化（NFV）：

- SDN 支持创建虚拟网络，用于隔离不同用户或服务的流量，提升网络安全性和灵活性。

5. 校园网络：

- 学校可以用 SDN 统一管理多个校区的网络，优化资源分配。

比如：在实际网络中的应用

假设你有一个公司的局域网，里面有许多计算机、打印机、服务器和路由器。每个设备之间通过交换机连接。传统网络中，交换机和路由器是独立工作的，它们的配置通常是固定的，不容易动态调整。而在 SDN 中：

- **控制器会监控整个网络**，了解每个设备的状态（比如交换机连接的端口、链路的带宽等）。
- **根据需求，控制器可以配置交换机**，例如让它们基于特定的规则（如目标 IP 地址、端口号等）转发流量。
- 如果网络中有大量的文件传输流量，控制器可能会动态调整流量路径，避免某些交换机或链路过载，确保网络性能。
- 如果交换机或链路发生故障，控制器可以快速感知并重新计算流量路径，确保网络能够继续运行。

Lab11.py 解释

1 概述

lab11.py 实现了一个基于 Ryu 控制器的 SDN 应用程序，其主要功能是一个简化的 OpenFlow 1.3 交换机，具体功能如下：

1. 流表管理：

- 控制器在接管交换机后，会安装一个默认的 **流表规则**，将所有未匹配的数据包转发到控制器（OFPP_CONTROLLER）。
- 当控制器收到未匹配的数据包后，会根据数据包的目标地址决定如何处理：
 - 如果目标 MAC 地址已经学习过（存在于 MAC 地址表中），控制器会下发对应的流表规则。
 - 如果目标地址未知，控制器会广播该数据包（OFPP_FLOOD）。

2. MAC 地址学习：

- 控制器通过监听交换机发送的数据包，学习源 MAC 地址与端口的对应关系，并将其存储在 **MAC 地址表** 中。
- 学习过程是动态的，能够适应网络拓扑的变化。

3. 流表超时设置：

- 添加的流表规则有 **空闲超时（idle_timeout）** 设置。如果一定时间内没有数据包匹配该规则，规则将被删除，避免流表占用过多资源。

4. 多协议支持：

- 代码对 **ARP、ICMP、TCP 和 UDP** 等常见协议进行了处理：
 - 不同协议类型（例如：TCP、P）的数据包会有不同的流表匹配规则。
 - 提供了灵活的匹配字段（如源 IP、目标 IP、源端口、目标端口等）。

2 问题解答：

为什么交换机连接到控制器时需要发送默认规则？

在 SDN 中，交换机（数据平面设备）本身并不做复杂的决策，所有的决策（例如，如何转发数据包）由控制器（控制平面）来决定。换句话说，交换机是“听话”的设备，控制器是“指挥官”。

默认规则的作用：

1. 捕获未匹配的流量：

- 当一个新的交换机连接到控制器时，它的流表是空的，意味着交换机不知道如何处理任何数据包。
- 为了避免丢弃所有数据包，控制器下发一条 **默认流表规则**：

```
match = parser.OFPMatch() # 匹配所有数据包
```

```
actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
ofproto.OFPCML_NO_BUFFER)] # 转发到控制器
```

- 这条规则告诉交换机：“所有未匹配的数据包都发送到控制器”。

2. 让控制器接管网络管理：

- 通过默认规则，控制器可以收到网络中的所有未知流量。
- 控制器可以分析这些数据包，决定如何配置交换机的流表，例如：
 - 安装新的流表规则，用于处理特定的流量。
 - 直接对数据包进行处理。

流表规则是什么？

流表规则 是 OpenFlow 交换机用来决定如何处理数据包的规则集合。它的核心是一个 **匹配条件** 和 **动作列表**。

流表规则的结构：

- **匹配条件 (Match Fields) :**
 - 指定哪些数据包匹配这条规则。例如：
 - 根据源 MAC 地址或目标 MAC 地址匹配。
 - 根据源 IP 地址或目标 IP 地址匹配。
 - 根据协议类型 (TCP、UDP、ARP 等) 匹配。
 - 如果某条规则的条件与数据包不匹配，交换机会继续检查下一条规则。
- **动作列表 (Actions) :**
 - 指定匹配的包应该如何处理。例如：
 - 转发到某个端口。
 - 丢弃数据包。
 - 将数据包发送到控制器。

默认规则的匹配和动作：

- 在这段代码中，默认规则是：

```
match = parser.OFPMatch() # 匹配所有数据包
actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
ofproto.OFPCML_NO_BUFFER)] # 动作：发送到控制器
```

- 匹配条件：**匹配所有数据包**（无条件匹配）。
- 动作：**发送到控制器**（将数据包发送到控制器处理）。

数据包是从哪里来的？

数据包的来源可以分为两种情况：

(1) 来自局域网设备

- 当交换机连接到局域网时，网络中的设备（如计算机、服务器）会向交换机发送数据包。

- 例如：
 - 计算机 A 想与计算机 B 通信，它会向交换机发送一个数据包。
 - 如果交换机中没有与计算机 B 相关的流表规则，它会根据默认规则将数据包发送到控制器。

(2) 来自控制器的测试或管理

- 控制器本身也可能发送数据包到交换机，测试网络状态或下发规则。
- 例如：
 - 控制器可以发送“探测包”到交换机，用于检测交换机的连通性或状态。

3 实现的具体功能

1. 默认规则的安装：

- 在交换机连接到控制器时，控制器会发送一个默认流表规则，匹配所有数据包并将其转发到控制器：

```
match = parser.OFPMatch()
actions = [parser.OFPACTIONOutput(ofproto.OFPP_CONTROLLER,
ofproto.OFPCML_NO_BUFFER)]
```

- 这确保了交换机在没有匹配规则时，会将数据包发送到控制器处理。

2. MAC 地址表学习：

- 控制器通过数据包中的源 MAC 地址和入端口，动态建立 MAC 地址到端口的映射表：

```
self.mac_to_port[dpid][src] = in_port
```

- 这让交换机可以根据目标 MAC 地址的学习记录，直接将数据包转发到正确的端口，而不是广播。

3. 数据包处理逻辑：

- 控制器根据目标 MAC 地址，决定如何处理数据包：
 - 如果目标地址已知（在 MAC 地址表中），控制器会创建对应的流表规则，并将数据包发送到指定端口。
 - 如果目标地址未知，数据包会被广播（Flood）到所有端口。

4. 流表规则的动态管理：

- 当目标 MAC 地址已知时，控制器会向交换机添加流表规则：

```
self.add_flow1(datapath, 1, match, actions)
```

- 规则匹配字段可能包括 MAC 地址、IP 地址、协议类型等。
- 超时时间设置为 5 秒（idle_timeout=5），使规则在一定时间内无数据匹配时自动删除。

5. 支持常见协议的流表匹配：

- 对于 IP 数据包，代码根据协议类型（ICMP、TCP、UDP）生成不同的流表规则。例如：

- **TCP 数据包：**

```
match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP,
in_port=in_port, ipv4_src=srcip, ipv4_dst=dstip,
ip_proto=protocol, tcp_src=t.src_port, tcp_dst=t.dst_port)
```

- **UDP 数据包：**

```
match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP,
in_port=in_port, ipv4_src=srcip, ipv4_dst=dstip,
ip_proto=protocol, udp_src=u.src_port, udp_dst=u.dst_port)
```

6 Packet-Out 消息

- **Packet-Out** 是 OpenFlow 协议中的一种消息类型。
- 当交换机将某个数据包发送到控制器后（Packet-In），控制器可以使用 Packet-Out 消息告诉交换机如何处理这个数据包。
- Packet-Out 消息可以指定：
 - 将数据包发送到哪个端口（或多个端口）。
 - 数据包的具体内容（可以是交换机接收的原始数据包，也可以是新构造的数据包）。

(1) 获取数据包内容

```
data = None
if msg.buffer_id == ofproto.OFP_NO_BUFFER:
    data = msg.data
```

- **msg.buffer_id:**

- 交换机中的数据包通常会被存储在缓存中，buffer_id 是交换机缓存区中该数据包的标识符。
- 如果 buffer_id == OFP_NO_BUFFER，表示数据包未被存储在交换机缓存中，此时数据包的完整内容会直接包含在 msg.data 中。

- **逻辑：**

- 如果没有缓存（OFP_NO_BUFFER），控制器直接从 msg.data 获取数据包内容。

(2) 构建 Packet-Out 消息

```
out = parser.OFPPacketOut(
    datapath=datapath,
    buffer_id=msg.buffer_id,
    in_port=in_port,
    actions=actions,
    data=data
)
```

- **OFPPacketOut:**

- 创建一条 Packet-Out 消息，告诉交换机如何处理这个数据包。
- 关键参数：

1. **datapath:**

- 表示与交换机的通信通道，用于标识该消息是发送给哪个交换机。

2. **buffer_id**:

- 如果交换机缓存了该数据包，控制器只需提供 **buffer_id**，交换机会直接取出缓存内容进行处理。
- 如果没有缓存 (**buffer_id == OFP_NO_BUFFER**)，需要提供完整的 **data**。

3. **in_port**:

- 指定数据包的输入端口，用于处理某些协议需要输入端口信息的情况。

4. **actions**:

- 指定数据包的转发动作，例如：转发到哪个端口。

5. **data**:

- 如果没有缓存，提供数据包的完整内容。

(3) 发送 Packet-Out 消息

```
datapath.send_msg(out)
```

- 将构建好的 Packet-Out 消息发送给交换机，执行指定的动作。

这段代码的主要作用是 **控制器指示交换机如何处理特定的数据包**。当交换机遇到没有匹配规则的数据包并将其发送给控制器后，控制器可以决定如何处理该数据包。

4 示例场景：如何工作？

假设网络中有三个设备（A、B 和 C），连接到同一交换机。

步骤 1：数据包进入交换机

- A 向 B 发送数据包，交换机中没有匹配规则，会将数据包转发到控制器（Packet-In 事件）。

步骤 2：控制器处理数据包

- 控制器检查数据包：
 - 学习到 A 的 MAC 地址，并将其与入端口绑定。
 - 检查 B 的 MAC 地址是否已学习：
 - 如果已知 B 的 MAC 地址，控制器下发流表规则，将数据包发送到对应端口。
 - 如果未知 B 的 MAC 地址，控制器广播数据包，B 响应后完成 MAC 地址学习。

步骤 3：安装流表规则

- 控制器为 A 到 B 的通信安装流表规则：
 - 流表规则匹配 A 的 MAC 地址和 B 的 MAC 地址，直接将后续数据包从 A 转发到 B。

InclassTest 3

1 实验操作

Terminal 1

```
xyuchern@xyuchernpc:~/Desktop/in3$ ryu-manager lab11.py
```

```
loading app lab11.py
```

```
loading app ryu.controller.ofp_handler
```

```
instantiating app lab11.py of SimpleSwitch13
```

```
instantiating app ryu.controller.ofp_handler of OFPHandler
```

`ryu-manager lab11.py` 这条命令的作用是 **启动 Ryu 控制器**，并加载 `lab11.py` 这个 Ryu 应用程序（即你实现的 SDN 控制器逻辑）。同时，`ryu-manager` 也会加载必要的默认模块（如 `ryu.controller.ofp_handler`）来处理 OpenFlow 消息。

loading app lab11.py

- Ryu 控制器正在加载你实现的应用程序 `lab11.py`。
- 这是你在之前编写的代码，其中定义了一个名为 `SimpleSwitch13` 的类，负责实现 SDN 控制器的逻辑（例如 MAC 地址学习、流表安装等）。

loading app ryu.controller.ofp_handler

- Ryu 控制器同时加载了 `ryu.controller.ofp_handler` 模块，这是 Ryu 的一个默认模块。
- **ofp_handler 的作用：**
 - 负责接收和处理来自交换机的 OpenFlow 消息（如 Packet-In、SwitchFeatures 等）。
 - 它是 Ryu 框架的核心组件，自动与 OpenFlow 交换机通信。

instantiating app lab11.py of SimpleSwitch13

- Ryu 控制器实例化了你定义的 `SimpleSwitch13` 类。
- 这意味着你的 SDN 应用程序已经被加载并开始运行。
- 这个类实现了核心逻辑，例如：
 - 接收交换机的 Packet-In 消息。
 - 根据数据包动态下发流表规则。

instantiating app ryu.controller.ofp_handler of OFPHandler

- Ryu 控制器实例化了默认的 **OFPHandler 类**，这是 `ryu.controller.ofp_handler` 模块的一部分。
- **作用：**作为 OpenFlow 协议的基础支持模块，负责监听、解析和响应 OpenFlow 消息。

现在发生了什么？

- 你的 Ryu 控制器已经启动并运行：
 - 它现在处于监听状态，等待 OpenFlow 交换机连接。
 - 一旦有交换机连接到控制器，控制器会开始接收交换机发送的消息，并根据 lab11.py 中的逻辑进行处理。
 - Ryu 控制器的默认行为：
 - 等待交换机连接（通过 TCP 端口 6633 或 6653，取决于配置）。
 - 交换机连接后，控制器会向交换机下发默认流表规则，接管交换机的流量管理。
-

Terminal 2

```
xyuchern@xyuchernpc:~$ sudo mn --controller=remote,ip=127.0.0.1,port=6653 --  
switch=ovsk,protocols=OpenFlow13
```

```
[sudo] password for xyuchern:
```

```
*** Creating network
```

```
*** Adding controller
```

```
*** Adding hosts:
```

```
h1 h2
```

```
*** Adding switches:
```

```
s1
```

```
*** Adding links:
```

```
(h1, s1) (h2, s1)
```

```
*** Configuring hosts
```

```
h1 h2
```

```
*** Starting controller
```

```
c0
```

```
*** Starting 1 switches
```

```
s1 ...
```

```
*** Starting CLI:
```

```
mininet>
```



```
sudo mn --controller=remote,ip=127.0.0.1,port=6653 --
switch=ovsk,protocols=OpenFlow13
```

- **--controller=remote,ip=127.0.0.1,port=6653:**
 - 指定 Mininet 中的交换机连接到一个远程 SDN 控制器。
 - `ip=127.0.0.1` 表示控制器运行在本地主机（你之前在另一个终端启动的 `ryu-manager`）。
 - `port=6653` 是 SDN 控制器监听的端口（你之前的 Ryu 控制器默认监听此端口）。
- **--switch=ovsk,protocols=OpenFlow13:**
 - 使用 Open vSwitch (OVS) 作为交换机的实现。
 - 指定使用 OpenFlow 1.3 协议，与控制器通信。
- **sudo mn:**
 - Mininet 是一个模拟器，可以创建虚拟网络拓扑，包括主机、交换机和控制器。
 - 需要 `sudo` 权限，因为它涉及到网络设备的创建。

*** Creating network

- Mininet 开始创建虚拟网络拓扑，包括主机、交换机和控制器。

*** Adding controller

- 添加一个远程控制器（remote），控制器的 IP 是 127.0.0.1，端口是 6653。

*** Adding hosts: h1 h2

- 添加两个虚拟主机，分别命名为 h1 和 h2。
- 这些主机在 Mininet 中模拟真实的计算机。

*** Adding switches: s1

- 添加一个虚拟交换机，命名为 s1。
- 这个交换机会按照 OpenFlow 1.3 协议与控制器通信。

*** Adding links: (h1, s1) (h2, s1)

- 创建了两条虚拟连接：
 - h1 连接到交换机 s1。
 - h2 连接到交换机 s1。

*** Configuring hosts h1 h2

- 为主机配置网络，例如分配 IP 地址。

*** Starting controller c0

- Mininet 中会自动启动一个虚拟控制器 c0，但是你指定了远程控制器（Ryu），因此这个虚拟控制器不会实际工作。

*** Starting 1 switches s1

- 启动了交换机 s1。
- 交换机会尝试连接到远程控制器 (127.0.0.1:6653)。

*** Starting CLI:

- 启动 Mininet 的命令行界面 (CLI)，你现在可以在 mininet> 提示符下输入命令，操控虚拟网络。

```
h1 --- s1 --- h2
      |
      Controller (127.0.0.1:6653)
```

- **h1** 和 **h2** 是两个虚拟主机。
- **s1** 是一个支持 OpenFlow 1.3 的交换机。
- **Controller** 是你运行的 Ryu 控制器 (lab11.py)，通过 OpenFlow 协议控制交换机。

我看明白了在运行这个代码时候，原来 termianl1 冒出来以下内容：

```
packet in 0000000000000001 26:00:b9:5f:ce:1b 33:33:00:00:00:16 1
packet in 0000000000000001 de:a0:c9:ab:c5:d5 33:33:00:00:00:16 2
packet in 0000000000000001 26:00:b9:5f:ce:1b 33:33:ff:5f:ce:1b 1
packet in 0000000000000001 26:00:b9:5f:ce:1b 33:33:00:00:00:16 1
```

日志的每一行记录了一个 **Packet-In** 消息，具体格式如下：

```
packet in <datapath ID> <源 MAC 地址> <目标 MAC 地址> <输入端口>
```

各字段的含义：

1. packet in:

- 表示这是一个 **Packet-In** 消息，意味着交换机无法处理某个数据包，因此将该数据包发送到控制器。
- 控制器会根据数据包内容做出决策，例如学习 MAC 地址或安装流表规则。

2. <datapath ID> (例子：0000000000000001)：

- 表示发送 Packet-In 消息的交换机的唯一标识符 (datapath ID)。
- 在你的拓扑中，0000000000000001 是交换机 s1 的 ID。

3. <源 MAC 地址> (例子：26:00:b9:5f:ce:1b 或 de:a0:c9:ab:c5:d5)：

- 数据包的源 MAC 地址，表示这是谁发送的数据包。
- 例如：
 - 26:00:b9:5f:ce:1b 是主机 h1 的 MAC 地址。
 - de:a0:c9:ab:c5:d5 是主机 h2 的 MAC 地址。

4. <目标 MAC 地址> (例子: 33:33:00:00:00:16 或 33:33:00:00:00:02) :

- 数据包的目标 MAC 地址, 表示数据包希望发往哪个设备。
- 33:33:... 形式的 MAC 地址通常是 **组播地址**, 例如:
 - 33:33:00:00:00:16 是 IPv6 多播地址。
 - 33:33:00:00:00:02 是另一个多播地址。

5. <输入端口> (例子: 1 或 2) :

- 表示交换机从哪个端口接收了数据包。
- 在你的网络中:
 - 1 是主机 h1 连接到交换机的端口。
 - 2 是主机 h2 连接到交换机的端口。

数据包的来源和处理流程:

1. 主机 h1 (26:00:b9:5f:ce:1b) 或 h2 (de:a0:c9:ab:c5:d5) 发送数据包。
2. 交换机 s1 收到数据包, 但它的流表中没有匹配规则。
3. 根据默认规则, 交换机将数据包发送到控制器。
4. 控制器通过日志记录下这些 Packet-In 消息。

Terminal 2

mininet> pingall

*** Ping: testing ping reachability

h1 -> h2

h2 -> h1

*** Results: 0% dropped (2/2 received)

mininet>

pingall

- pingall 是 Mininet 中的命令, 用来测试网络中所有主机之间的连通性。
- 它的作用是让每台主机向其他主机发送 ICMP Ping 数据包, 检查数据包是否能够成功到达目标。

*** Ping: testing ping reachability

h1 -> h2

h2 -> h1

*** Results: 0% dropped (2/2 received)

1. h1 -> h2 和 h2 -> h1:

- 表示主机 h1 能成功 Ping 通主机 h2, 主机 h2 也能成功 Ping 通主机 h1。
- 这说明主机之间的通信是正常的。

2. *** Results: 0% dropped (2/2 received):

- 0% dropped 表示所有的 Ping 数据包都成功到达目标，没有丢失。
- 2/2 received 表示总共发送了 2 个数据包 (h1 -> h2 和 h2 -> h1)，目标主机全部收到了。

为什么这是一个重要的测试？

这表明你的 SDN 网络拓扑和控制器逻辑都正常工作：

1. 交换机与控制器的通信正常：

- 交换机成功向控制器发送 Packet-In 消息。
- 控制器分析了 Packet-In 消息并下发了流表规则，指导交换机如何转发后续数据包。

2. 流表规则被正确安装：

- 在第一次 h1 -> h2 的 Ping 时，交换机会向控制器发送 Packet-In 消息。
- 控制器为 h1 -> h2 和 h2 -> h1 的通信下发了流表规则。
- 后续数据包直接按照流表规则在交换机上转发，不再需要控制器干预。

3. 主机之间的通信正常：

- 测试证明主机之间可以正常发送和接收数据包，网络功能如预期运行。

Packet-In 和流表的关系

当你运行 pingall 时，网络的实际工作流程如下：

第一次 Ping (h1 -> h2)：

1. 主机 h1 向 h2 发送 ICMP Echo 请求 (Ping 数据包)。
2. 交换机 s1 收到数据包，发现流表中没有匹配规则，于是触发 Packet-In 消息，将数据包发送到控制器。
3. 控制器分析数据包，发现：
 - 源 MAC 地址：h1
 - 目标 MAC 地址：h2
4. 控制器学习 MAC 地址，并下发流表规则：
 - 匹配 h1 -> h2 的流量，转发到端口 2 (h2 的连接端口)。
 - 匹配 h2 -> h1 的流量，转发到端口 1 (h1 的连接端口)。
5. 交换机将数据包转发到 h2。

第二次 Ping (h2 -> h1)：

- 由于控制器已经为 h2 -> h1 安装了流表规则，交换机直接根据流表规则转发数据包，而无需再发送 Packet-In 消息。
-

Terminal 3

```
xyuchern@xyuchernpc:~$ sudo ovs-ofctl dump-flows s1 -O OpenFlow13
```

[sudo] password for xyuchern:

```
cookie=0x0, duration=5403.710s, table=0, n_packets=37, n_bytes=2754, priority=0
actions=CONTROLLER:65535
```

```
1 sudo ovs-ofctl dump-flows s1 -O OpenFlow13
```

命令: **ovs-ofctl dump-flows**

- **ovs-ofctl** 是用于管理 Open vSwitch 的命令行工具。
- **dump-flows** 用于查看交换机中已安装的 OpenFlow 流表规则。

选项: **s1**

- 指定要查询的交换机是 **s1**。

选项: **-O OpenFlow13**

- 指定使用 **OpenFlow 1.3** 协议进行交互。
- 因为你在运行 Mininet 时选择了 OpenFlow 1.3 协议，所以需要这个选项。

```
cookie=0x0, duration=5403.710s, table=0, n_packets=37, n_bytes=2754, priority=0
actions=CONTROLLER:65535
```

字段解析:

1. **cookie=0x0:**

- 流表规则的标识符，默认值是 **0x0**。
- 它可以用于标记不同的规则，但在你的代码中没有使用 cookie，所以显示为 **0x0**。

2. **duration=5403.710s:**

- 这条规则在交换机中已经存在的时间（以秒为单位）。
- 表示这条规则已存在 5403.71 秒（约 1.5 小时）。

3. **table=0:**

- 表示这条规则属于 **表 0**。
- OpenFlow 交换机可以有多个流表（例如表 0、表 1、表 2 等），每张表可以有不同的用途。默认规则一般位于表 0。

4. **n_packets=37:**

- 统计匹配这条规则的数据包数量。
- 表示在这段时间内，共有 37 个数据包匹配了这条规则。

5. **n_bytes=2754:**

- 统计匹配这条规则的数据总字节数。
- 表示匹配这条规则的数据包总大小是 2754 字节。

6. **priority=0:**

- 这条规则的优先级为 0，表示这是最低优先级规则。
- 低优先级规则通常是默认规则（catch-all），用于处理未匹配任何特定规则的流量。

7. **actions=CONTROLLER:65535:**

- **动作（actions）：**
 - 这条规则的动作是将匹配到的数据包发送到控制器。
 - **CONTROLLER:65535** 表示数据包会被发送到控制器，并携带最多 65535 字节的数据包内容（即完整数据包）。
- 这是你在 **lab11.py** 中设置的默认规则。

这是一条 **默认流表规则**，设置为最低优先级，用于捕获所有未匹配其他规则的数据包。

具体作用：

1. 匹配所有流量：

- 因为优先级是 0，且没有设置具体的匹配条件，这条规则会匹配所有未被其他规则匹配的数据包。

2. 发送到控制器：

- 动作是将数据包发送到控制器（Packet-In 消息）。
- 控制器会收到这些数据包，分析内容并决定如何处理：
 - 学习 MAC 地址。
 - 下发新的流表规则。

如何验证是否添加了其他规则？

目前你只看到了一条默认规则。如果控制器通过 **lab11.py** 逻辑安装了新的规则，你应该会看到类似这样的输出：

```
cookie=0x0, duration=10.340s, table=0, n_packets=5, n_bytes=420,
priority=1,ip,in_port=1,actions=output:2
cookie=0x0, duration=10.200s, table=0, n_packets=3, n_bytes=280,
priority=1,ip,in_port=2,actions=output:1
```

你的输出表明：

- 当前交换机中只有一条默认规则，用于将未匹配流量发送到控制器。
- 这是控制器接管交换机后的初始状态，确保所有未知流量都会通过 **Packet-In** 消息发送到控制器进行处理。

After pingall in terminal 2 Then

Terminal 3

```
xyuchern@xyuchernpc:~$ sudo ovs-ofctl dump-flows s1 -O OpenFlow13
```

```
cookie=0x0, duration=4.168s, table=0, n_packets=3, n_bytes=294, idle_timeout=5,
priority=1,icmp,in_port="s1-eth1",nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=output:"s1-eth2"
```

```
cookie=0x0, duration=4.162s, table=0, n_packets=3, n_bytes=294, idle_timeout=5,
priority=1,icmp,in_port="s1-eth2",nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=output:"s1-eth1"
```

```
cookie=0x0, duration=6002.464s, table=0, n_packets=45, n_bytes=3426, priority=0
actions=CONTROLLER:65535
```

```
cookie=0x0, duration=4.168s, table=0, n_packets=3, n_bytes=294, idle_timeout=5,
priority=1,icmp,in_port="s1-eth1",nw_src=10.0.0.1,nw_dst=10.0.0.2
actions=output:"s1-eth2"
```

1. **cookie=0x0:**

- 这是流表规则的标识符，用于标记不同的规则。
- 默认值是 0x0。

2. **duration=4.168s:**

- 这条规则已经存在了 4.168 秒。

3. **table=0:**

- 规则属于表 0，这是 OpenFlow 中的默认流表。

4. **n_packets=3, n_bytes=294:**

- 已有 3 个数据包匹配了这条规则，累计数据量为 294 字节。
- 这些数据包是 h1 (10.0.0.1) 发送到 h2 (10.0.0.2) 的 Ping 请求。

5. **idle_timeout=5:**

- 如果 5 秒内没有新的数据包匹配该规则，规则将自动被移除。

6. **priority=1:**

- 规则优先级为 1，比默认规则（优先级 0）高。

7. **icmp,in_port="s1-eth1",nw_src=10.0.0.1,nw_dst=10.0.0.2:**

- 匹配条件：
 - 协议类型：icmp（Ping 请求使用 ICMP 协议）。
 - 输入端口：s1-eth1，即 h1 连接的交换机端口。
 - 源 IP 地址：10.0.0.1，h1 的 IP 地址。
 - 目标 IP 地址：10.0.0.2，h2 的 IP 地址。

8. actions=output:"s1-eth2":

- 动作是将数据包转发到端口 **s1-eth2**，即连接 **h2** 的端口。

```
cookie=0x0, duration=4.162s, table=0, n_packets=3, n_bytes=294, idle_timeout=5,
priority=1, icmp, in_port="s1-eth2", nw_src=10.0.0.2, nw_dst=10.0.0.1
actions=output:"s1-eth1"
```

这条规则是为 **h2** 到 **h1** 的 Ping 响应安装的流表，字段解析与上一条类似：

- 输入端口是 **s1-eth2**，源 IP 是 **10.0.0.2**，目标 IP 是 **10.0.0.1**。
- 动作是将数据包转发到 **s1-eth1**，即连接 **h1** 的端口。

```
cookie=0x0, duration=6002.464s, table=0, n_packets=45, n_bytes=3426, priority=0
actions=CONTROLLER:65535
```

这是之前的默认规则（优先级 0），用于处理未匹配任何特定规则的数据包。它的作用是将未匹配的流量发送到控制器（Packet-In）。

通过你的操作和输出，我们可以总结出控制器的行为流程：

1. 第一次 Ping（Packet-In 触发）：

- 当 **h1** 向 **h2** 发送 Ping 请求时，交换机 **s1** 没有匹配规则，因此触发 Packet-In 消息，将数据包发送到控制器。
- 控制器分析数据包，学习 MAC 地址，并为 **h1 -> h2** 和 **h2 -> h1** 安装流表规则。

2. 流表规则下发：

- 控制器下发两条规则：
 - 一条规则将 **h1 -> h2** 的流量转发到 **s1-eth2**。
 - 一条规则将 **h2 -> h1** 的流量转发到 **s1-eth1**。

3. 后续 Ping（直接匹配流表）：

- 后续的 Ping 请求和响应直接匹配流表规则，在交换机中完成转发，不再需要控制器干预。
- 这提高了网络效率，因为数据包不再需要经过控制器。

你的网络拓扑如下：

h1 (10.0.0.1) --- s1 --- h2 (10.0.0.2)

- 主机 **h1** 和 **h2** 可以通过交换机 **s1** 直接通信。
 - 交换机 **s1** 中有三条流表规则：
 1. 默认规则：匹配未定义流量，发送到控制器。
 2. 从 **h1** 到 **h2** 的规则。
 3. 从 **h2** 到 **h1** 的规则。
-

Terminal 2

```
mininet> h1 hping3 h2 -c 10000 -S --flood --rand-source -V
```

using h1-eth0, addr: 10.0.0.1, MTU: 1500

HPING 10.0.0.2 (h1-eth0 10.0.0.2): S set, 40 headers + 0 data bytes

hping in flood mode, no replies will be shown

```
h1 hping3 h2 -c 10000 -S --flood --rand-source -V
```

这是使用 **hping3** 工具从主机 **h1** 向主机 **h2** 发送一个大规模的模拟攻击（Flooding Attack）。以下是详细解析：

基本语法：h1 hping3 h2 ...

- 表示在 Mininet 中从主机 h1 执行 hping3 命令，目标是主机 h2。
- hping3 是一个网络工具，用于生成自定义的网络数据包，常用于测试、防火墙检查或模拟攻击。

具体参数：

1. -c 10000:

- 指定发送数据包的数量。
- 在这个命令中，你将发送 **10000 个数据包**。

2. -S:

- 表示设置 TCP SYN 标志位。
- 生成的是 TCP SYN 数据包，模拟三次握手的第一个阶段，常用于 **SYN Flood 攻击**。

3. --flood:

- 启用 **Flood 模式**，以尽可能快的速度发送数据包。
- 此模式会忽略响应，因此不会显示目标主机的回复。

4. --rand-source:

- 随机生成源 IP 地址。
- 这会使每个数据包看起来像是从不同的 IP 地址发送，模拟分布式攻击（DDoS）。

5. -V:

- 启用详细模式（Verbose），输出发送数据包的信息。

```
using h1-eth0, addr: 10.0.0.1, MTU: 1500
HPING 10.0.0.2 (h1-eth0 10.0.0.2): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
```

各部分解析：

1. **using h1-eth0, addr: 10.0.0.1, MTU: 1500:**

- **h1-eth0** 是主机 h1 的网络接口，IP 地址为 10.0.0.1。
- **MTU: 1500** 表示数据包的最大传输单元为 1500 字节。

2. **HPING 10.0.0.2 (h1-eth0 10.0.0.2): S set, 40 headers + 0 data bytes:**

- 目标是主机 h2 的 IP 地址 (10.0.0.2)。
- **S set** 表示数据包设置了 TCP SYN 标志位。
- 每个数据包包含 **40 字节的 TCP/IP 头部**，不包含额外的数据负载。

3. **hping in flood mode, no replies will be shown:**

- **Flood 模式启用**，表示数据包将以最高速率发送。
- **不会显示目标主机的响应信息。**

实际含义：你正在模拟 SYN Flood 攻击

- **SYN Flood** 是一种常见的 **DoS 攻击 (Denial of Service)**，通过向目标主机发送大量的 TCP SYN 请求，耗尽目标的资源。
- 你的命令还使用了 **--rand-source**，模拟了分布式 DoS (DDoS) 攻击：
 - 每个 SYN 数据包的源 IP 地址是随机生成的，使得目标主机难以追踪或过滤。

h1 (10.0.0.1) --- s1 --- h2 (10.0.0.2)

- 主机 **h1** 向主机 **h2** 发送大量伪造的 SYN 数据包。
- 交换机 **s1** 将这些数据包转发到目标主机 **h2**，并且流量会通过 SDN 控制器管理的流表规则处理。

可能的影响：

1. 对主机 **h2** 的影响：

- 如果 **h2** 没有防御机制，它会尝试响应每个伪造的 SYN 请求（发送 SYN-ACK），导致资源耗尽（如 CPU 或内存）。
- 随着攻击流量增加，**h2** 可能无法处理其他正常请求。

2. 对控制器和交换机的影响：

- 初始时，交换机可能将部分数据包发送到控制器（Packet-In），增加控制器的负担。
- 如果控制器安装了流表规则，交换机会直接处理这些数据包，可能会导致交换机端口流量激增。

此时运行 Terminal 3 : `sudo ovs-ofctl dump-flows s1 -O OpenFlow13`

```
cookie=0x0, duration=0.033s, table=0, n_packets=0, n_bytes=0, idle_timeout=5,
priority=1,tcp,in_port="s1-
eth1",nw_src=161.73.187.23,nw_dst=10.0.0.2,tp_src=21217,tp_dst=0 actions=output:"s1-
eth2"
```

```
cookie=0x0, duration=0.033s, table=0, n_packets=0, n_bytes=0, idle_timeout=5,
priority=1,tcp,in_port="s1-
eth1",nw_src=37.108.15.109,nw_dst=10.0.0.2,tp_src=21218,tp_dst=0 actions=output:"s1-
eth2"
```

```
cookie=0x0, duration=0.033s, table=0, n_packets=0, n_bytes=0, idle_timeout=5,
priority=1,tcp,in_port="s1-
eth1",nw_src=51.56.199.91,nw_dst=10.0.0.2,tp_src=21219,tp_dst=0 actions=output:"s1-eth2"
```

```
cookie=0x0, duration=0.032s, table=0, n_packets=0, n_bytes=0, idle_timeout=5,
priority=1,tcp,in_port="s1-
eth1",nw_src=225.137.73.161,nw_dst=10.0.0.2,tp_src=21221,tp_dst=0 actions=output:"s1-
eth2"
```

```
cookie=0x0, duration=0.032s, table=0, n_packets=0, n_bytes=0, idle_timeout=5,
priority=1,tcp,in_port="s1-
eth1",nw_src=239.104.250.22,nw_dst=10.0.0.2,tp_src=21222,tp_dst=0 actions=output:"s1-
eth2"
```

```
cookie=0x0, duration=0.032s, table=0, n_packets=0, n_bytes=0, idle_timeout=5,
priority=1,tcp,in_port="s1-
eth1",nw_src=191.74.100.166,nw_dst=10.0.0.2,tp_src=21220,tp_dst=0 actions=output:"s1-
eth2"
```

.....

```
sudo ovs-ofctl dump-flows s1 -O OpenFlow13
```

它列出了交换机 `s1` 中当前的流表规则。这些规则是由于你之前在 Mininet 中运行了 `h1 hping3 h2 ...` 命令所触发的。让我们解析这些规则并理解它们的含义。

```
cookie=0x0, duration=0.033s, table=0, n_packets=0, n_bytes=0, idle_timeout=5,
priority=1,tcp,in_port="s1-
eth1",nw_src=161.73.187.23,nw_dst=10.0.0.2,tp_src=21217,tp_dst=0
actions=output:"s1-eth2"
```

1. **cookie=0x0:**

- 流表规则的标识符，用于标记规则来源。
- 在你的实验中，所有规则的 `cookie` 都是默认值 `0x0`。

2. **duration=0.033s:**

- 这条规则存在的时间。这里的值是 **0.033** 秒，说明这条规则刚刚被添加。

3. table=0:

- 规则属于表 0，这是默认的 OpenFlow 流表。

4. n_packets=0, n_bytes=0:

- 到目前为止，没有任何数据包匹配这条规则 (**n_packets=0**)。
- 数据量为 0 字节 (**n_bytes=0**)。

5. idle_timeout=5:

- 如果 5 秒内没有新的数据包匹配此规则，规则将被移除。

6. priority=1:

- 规则优先级为 1，比默认规则（优先级 0）高。
- 这意味着当数据包匹配这些规则时，会优先处理这些规则。

7. tcp:

- 匹配条件之一，表示规则适用于 TCP 数据包。

8. in_port="s1-eth1":

- 匹配条件之一，表示规则适用于从端口 **s1-eth1**（连接到主机 **h1**）进入的数据包。

9. nw_src=161.73.187.23:

- 匹配条件之一，表示规则适用于源 IP 地址为 **161.73.187.23** 的数据包。
- 这些源 IP 地址是由于你使用了 **--rand-source** 参数，生成的随机伪造 IP。

10. nw_dst=10.0.0.2:

- 匹配条件之一，表示规则适用于目标 IP 地址为 **10.0.0.2** 的数据包（主机 **h2** 的 IP 地址）。

11. tp_src=21217, tp_dst=0:

- 匹配条件之一：
 - **tp_src=21217**: 表示数据包的源端口是 21217。
 - **tp_dst=0**: 表示数据包的目标端口未指定。

12. actions=output:"s1-eth2":

- 动作：将匹配的数据包转发到端口 **s1-eth2**（连接到主机 **h2**）。

这些规则是由你的 Ryu 控制器动态生成并下发的。当你运行 **h1 hping3 h2 ...** 命令后：

1. 主机 **h1** 开始向 **h2** 发送大量 TCP SYN 数据包。
2. 由于你在 **hping3** 命令中使用了 **--rand-source**，每个数据包的源 IP 地址和源端口都是随机生成的。
3. 交换机 **s1** 收到这些数据包，但流表中没有匹配规则，因此将数据包发送到控制器（Packet-In 消息）。
4. 控制器分析这些数据包，为每个源 IP 地址生成了一条对应的流表规则，并下发给交换机。
 - 这些规则将匹配相应的源 IP 地址、目标 IP 地址、源端口等字段。
 - 匹配的数据包会被转发到端口 **s1-eth2**，到达主机 **h2**。

可能的影响

(1) 交换机的流表资源耗尽

- OpenFlow 交换机的流表容量是有限的。如果流表规则过多（例如每个数据包都需要新规则），会导致交换机流表资源耗尽。
- 当流表资源耗尽时，新的数据包将无法匹配流表规则，性能会大幅下降。

(2) 控制器的负担增加

- 每个未匹配的数据包都会触发 Packet-In 消息，发送到控制器。
- 如果数据包数量非常大，控制器可能会因为处理大量 Packet-In 消息而变得过载。

(3) 网络性能下降

- 交换机和控制器的资源消耗会导致正常流量的处理效率降低，甚至可能导致网络中断。

Terminal 2

```
mininet> h1 hping3 h2 -c 10000 -S --flood --rand-source -V
using h1-eth0, addr: 10.0.0.1, MTU: 1500
HPING 10.0.0.2 (h1-eth0 10.0.0.2): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 10.0.0.2 hping statistic ---
203321331 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=5.15 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.350 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.076 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.086 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.083 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.169 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.117 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.124 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.077 ms
```

ctrl +c 后

```
--- 10.0.0.2 hping statistic ---  
203321331 packets transmitted, 0 packets received, 100% packet loss  
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

1. 203321331 packets transmitted:

- 表示 hping3 共发送了 203,321,331 个数据包，这是由于 --flood 模式的高频发送造成的。

2. 0 packets received, 100% packet loss:

- 表示目标主机 h2 没有回复任何 SYN-ACK 数据包。
- 可能的原因：
 - 主机 h2 被攻击流量压垮，无法处理正常请求。
 - 或者由于 hping3 的设计，在 --flood 模式下不显示回复。

3. round-trip min/avg/max = 0.0/0.0/0.0 ms:

- 因为 --flood 模式只关注数据包的发送，不记录 RTT（往返时间）。

```
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=5.15 ms  
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.350 ms  
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.076 ms  
...
```

1. 正常通信恢复:

- 主机 h1 可以成功 Ping 通 h2。
- 说明你停止 Flooding 攻击后，网络恢复了正常。

2. RTT（往返时间）:

- 第一条 Ping 的延迟较高（5.15 ms），可能是由于网络刚刚恢复，流表规则被重新安装。
- 随后的延迟非常低（约 0.1 ms），说明网络负载较低，通信效率恢复正常。

Flooding 攻击的影响

1. 控制器和交换机的负载:

- 攻击期间，交换机 s1 和控制器需要处理大量随机源 IP 数据包：
 - 交换机：为每个伪造 IP 地址安装新的流表规则，流表可能迅速膨胀。
 - 控制器：处理大量 Packet-In 消息，可能导致延迟或过载。

2. 主机 h2 的资源消耗:

- 主机 h2 收到大量伪造的 SYN 数据包，但由于 --rand-source 随机生成源 IP，无法完成 TCP 三次握手，导致资源被浪费。

3. 正常流量被影响:

- 攻击期间，正常流量可能受到阻塞或延迟，Ping 测试无法进行。

- 攻击结束后，网络恢复正常。

我发现如果不等待一忽而 ping 会出现如下状况:

```
mininet> h1 hping3 h2 -c 10000 -S --flood --rand-source -V
using h1-eth0, addr: 10.0.0.1, MTU: 1500
HPING 10.0.0.2 (h1-eth0 10.0.0.2): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 10.0.0.2 hping statistic ---
396683 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
^C
--- 10.0.0.2 ping statistics ---
7 packets transmitted, 0 received, 100% packet loss, time 6144ms
```

- **Flooding 攻击 (hping3) 停止后**，你立即尝试用 ping 测试主机 h1 和 h2 的连通性，结果显示：
 - **100% 数据包丢失 (0 received)。**
 - 即使攻击已停止，h1 和 h2 仍然无法通信。

可能原因分析

(1) 流表未超时，阻塞正常流量

- 在 Flooding 攻击期间，Ryu 控制器为每个伪造的源 IP 地址下发了单独的流表规则。
- 这些规则可能优先级较高，但由于攻击结束后没有新伪造流量匹配它们，这些规则不会立即被清除。
- 如果这些流表规则与正常 Ping 流量的规则冲突，可能会导致正常流量无法匹配合适的规则，最终被丢弃。

(2) 交换机和控制器的缓冲区尚未恢复

- 攻击期间，交换机和控制器处理了大量的 Packet-In 消息，可能导致内部缓冲区积压或超载。

- 即使攻击停止，缓冲区需要一定时间清空并恢复正常。

(3) 主机的 TCP/IP 堆栈受到影响

- 如果主机 h2 在攻击期间收到大量伪造的 SYN 数据包，其 TCP/IP 堆栈可能因为资源耗尽（如连接表溢出）而无法立即恢复处理新连接。

(4) ICMP 流表规则未及时更新

- 如果之前的流表规则过多，控制器可能无法及时为新的 ICMP Ping 请求下发规则，导致 Ping 流量被丢弃。

检查交换机流表

在 **Terminal 3** 中，运行以下命令查看流表规则：

```
sudo ovs-ofctl dump-flows s1 -O OpenFlow13
```

- 观察点：
 - 检查流表中是否仍有大量与伪造 IP 地址相关的规则。
 - 检查是否存在匹配正常 ICMP 流量的规则（如 icmp 和 nw_src=10.0.0.1, nw_dst=10.0.0.2）。

这个任务要求你在一个软件定义网络（SDN）的环境中执行一个 **拒绝服务攻击 (DoS)**，目标是网络交换机的**流表 (flow table)**。以下是任务的具体解析：

任务解读

1. 执行 DoS 攻击：

- 你需要模拟或执行一种 DoS 攻击，目标是让交换机的流表资源被耗尽。
- **SDN 中的流表存储了交换机用来转发流量的规则，但它的容量有限。如果攻击者用大量伪造的流量填满流表，合法流量就无法处理，导致网络性能下降或服务中断。**

2. 识别安全漏洞：

- 攻击的目的是让你理解这种攻击是如何利用 SDN 架构中的安全漏洞的。
- 比如，SDN 的控制平面与数据平面分离，这使得控制平面可能因过载而被攻击。

3. 提出缓解方案：

- 你需要思考如何防止类似的攻击。提出一种有效的解决方案，例如：
 - **流表容量管理**：限制每个源 IP 可以占用的流表条目数。
 - **流量速率限制**：在控制器或交换机层面，对未知流量进行速率限制。
 - **攻击检测与响应**：利用入侵检测系统（IDS）快速识别并阻止恶意流量。
-

背景知识补充

1. SDN 架构：

- SDN 将网络控制平面与数据平面分离，网络流量通过**控制器**动态编程。
- 交换机的流表由控制器安装，用于匹配和转发数据包。

2. 流表的作用：

- 流表存储了转发规则，定义了如何处理特定的数据流。
- 容量有限，流表满了会导致合法流量无法安装规则，影响网络服务。

3. DoS 攻击原理：

- 攻击者通过发送大量伪造的数据包（如带有随机 IP 地址的流量），迫使交换机不断向控制器请求新规则，直到流表资源耗尽。