实验四：域内路由选择

在本实验中，你将利用网络模拟（Mininet）和抓包（Wireshark）工具，来模拟实现一个简单的路由器，并观察其域内路由行为。

一、实验目的

1. 理解和掌握ARP与ICMP协议在IPv4分组收发处理过程中的作用
2. 理解和掌握路由器静态路由转发过程
3. 理解和掌握动态路由选择协议的基本原理与过程
4. 提升文档阅读能力
5. 提升代码编写能力

二、实验内容

本实验包含以下三个任务：

- 任务一：使用Wireshark在全网范围内观察和分析ARP和ICMP协议。
- 任务二：实现一个具有静态路由功能的路由器。
- 任务三：在任务二的基础上，在路由器中实现动态路由协议RIP。

三、实验环境

本实验所需实验环境均已打包进虚拟机镜像中，可从以下地址进行下载和安装。

- 厦大云盘
- 腾讯微云

虚拟机中包含以下系统及软件：

1. **Ubuntu20.04**：操作系统系统用户 `vboxuser` ，密码 `ubuntu` 。
2. **Mininet2.3.1b4**：Mininet是一个网络仿真工具，允许在单机上构建网络拓扑，包括主机，交换机，SDN控制器。
3. **POX**：POX是一个OpenFlow的控制器，它可以控制Mininet中路由器的路由决策。在本实验中，我们将路由器接收到的每一个数据包都转发给POX控制器，POX控制器接着调用你实现的路由程序对数据包进行处理。
4. **Wireshark4.0.6**: 网络抓包和协议分析工具

四、实验流程

4.1 任务一：观察和分析ARP和ICMP协议过程

在实验二中，我们已学习了Wireshark的使用方法，并在发送主机侧观察了ARP及ICMP协议的具体过程，在本任务中，你将继续使用Wireshark在全网范围内（包括发送主机侧，转发路由器上，以及接收主机侧）观察并分析ARP、ICMP协议。

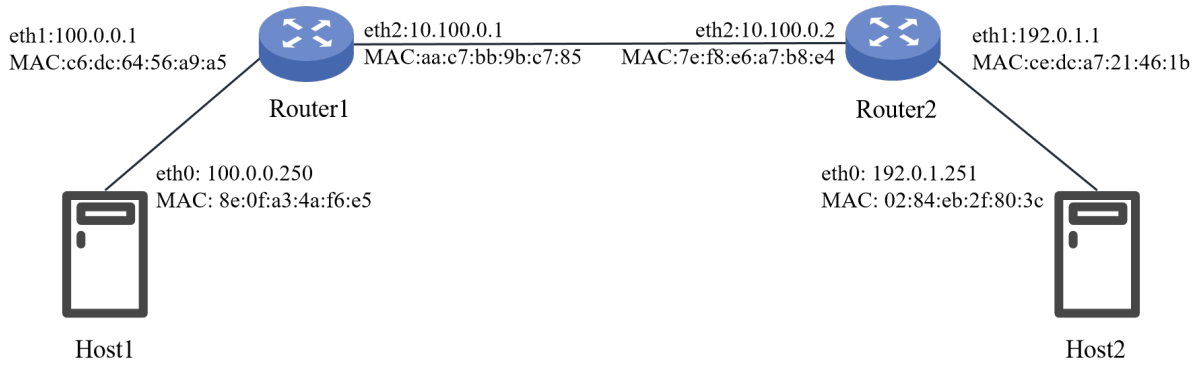
4.1.1 任务要求

请基于本任务提供的网络拓扑，

1. 追踪并分析在ping命令运行过程中，网络中各节点相关协议数据包的流向和变化过程。
2. 追踪并分析在traceroute命令运行过程中，网络中各节点相关协议数据包的流向和变化过程。
3. 列出各协议对应的以太网帧，在网络中经过各节点时的MAC地址变化。

4.1.2 任务拓扑

本任务网络拓扑及配置如下，该拓扑中涉及两台（host1和host2）主机，两台路由器（router1和router2）。



4.1.3 任务流程

1、进入虚拟机并定位到任务一代码目录：

```
/home/vboxuser/workspace/lab1
```

2、运行lab1.py脚本，启动给定的网络拓扑，你将看到如下输出。

```
vboxuser@ubuntu20:~/workspace/test/lab1$ sudo python lab1.py
[sudo] password for vboxuser:
*** Creating network
*** Adding controller
*** Adding hosts:
host1 host2 router1 router2
*** Adding switches:
*** Adding links:
(host1, router1) (host2, router2) (router1, router2)
*** Configuring hosts
host1 host2 router1 router2
*** Starting controller
c0
*** Starting 0 switches
*** Starting CLI:
mininet>
```

3、 在网络中每个节点上都运行Wireshark进行抓包。

```
mininet> host1 wireshark & > host1.log
mininet> host2 wireshark & > host2.log
mininet> router1 wireshark & > router1.log
mininet> router2 wireshark & > router2.log
```

4、 在Mininet终端中使用ping命令来测试host1到host2的网络连通性。追踪并分析在ping命令运行过程中，网络中各节点相关协议数据包的流向和变化过程。

```
mininet> host1 ping -c 3 host2
PING 192.0.1.251 (192.0.1.251) 56(84) bytes of data.
64 bytes from 192.0.1.251: icmp_seq=1 ttl=62 time=2.38 ms
64 bytes from 192.0.1.251: icmp_seq=2 ttl=62 time=0.586 ms
64 bytes from 192.0.1.251: icmp_seq=3 ttl=62 time=0.080 ms

--- 192.0.1.251 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2025ms
rtt min/avg/max/mdev = 0.080/1.014/2.377/0.985 ms
```

5、 断开router1和router2之间的链路后，再使用ping命令测host1到host2连通性，追踪并分析网络中各节点相关协议数据包的流向和变化过程。

```
mininet> link router1 router2 down
mininet> host1 ping -c 3 host2
PING 192.0.1.251 (192.0.1.251) 56(84) bytes of data.
From 100.0.0.1 icmp_seq=1 Destination Net Unreachable
From 100.0.0.1 icmp_seq=2 Destination Net Unreachable
From 100.0.0.1 icmp_seq=3 Destination Net Unreachable

--- 192.0.1.251 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2
053ms
```

6、 在Mininet终端运行traceroute命令来测试host1到host2的路由。追踪并分析在traceroute命令运行过程中，网络中各节点相关协议数据包的流向和变化过程。

```
mininet> host1 traceroute host2
traceroute to 192.0.1.251 (192.0.1.251), 30 hops max, 60 byte packets
 1  100.0.0.1 (100.0.0.1)  1.430 ms  1.351 ms  1.348 ms
 2  10.100.0.2 (10.100.0.2)  1.362 ms  1.367 ms  1.327 ms
 3  192.0.1.251 (192.0.1.251)  2.558 ms  2.568 ms  2.590 ms
```

4.2 任务二：静态路由

在该任务中，你将实现一个具有静态路由功能的路由器。该路由器将会接收以太网帧，并像真正的路由器一样对IPv4分组进行收发和处理，然后将它们转发到正确的端口。本任务已预先创建好了一个网络拓扑，并提供了具体的代码实现框架，保证路由器可正确地接收到以太网帧，进而使得你的主要精力可放在静态路由实现上。为了降低任务的复杂程度，绝大部分静态路由代码已给出，且代码中包含了详细的注释。

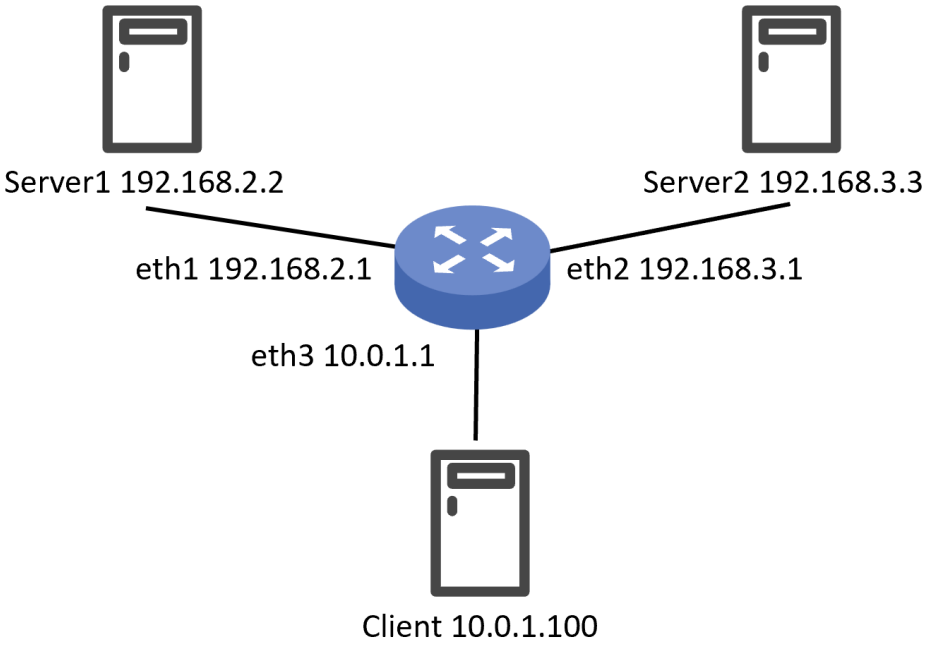
4.2.1 任务要求

请基于给定的拓扑及代码框架

1. 将静态路由代码补充完整，正确实现路由器的静态转发功能。
2. 阅读代码，画出路由器的代码流程图。

4.2.2 实验拓扑

本任务的网络拓扑包含一台客户端（client）主机，一台需要你实现静态路由功能的路由器，两台服务器（server1和server2）主机，三台主机通过路由器连接在一起，具体的IP和端口设置如下图所示。



4.2.3 实验流程

1、进入虚拟机并定位到具体的实验代码目录：

```
/home/vboxuser/workspace/lab2
```

2、进入对应的实验目录，并运行以下命令启动Mininet模拟器和POX控制器（POX控制器和Mininet需要分别运行在两个终端中）：

```
cd ~/workspace/lab2/
./run_pox.sh
./run_mininet.sh
```

运行POX控制器时，应能看到如下输出：

```
vboxuser@ubuntu20:~/workspace/lab2$ ./run_pox.sh
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:.home.vboxuser.workspace.cs144_lab3.pox_module.cs144.srhandler:cr
eated server
INFO:core:POX 0.2.0 (carp) is up.
```

运行Mininet模拟器时，应能看到如下输出：

```
vboxuser@ubuntu20:~/workspace/lab2$ ./run_mininet.sh
*** Shutting down stale SimpleHTTPServers
*** Shutting down stale webservers
server1 192.168.2.2
server2 192.168.3.3
client 10.0.1.100
sw0-eth1 192.168.2.1
sw0-eth2 192.168.3.1
sw0-eth3 10.0.1.1
*** Successfully loaded ip settings for hosts
 {'server1': '192.168.2.2', 'sw0-eth3': '10.0.1.1', 'sw0-eth1': '192.168
*** Creating network
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
client server1 server2
*** Adding switches:
sw0
*** Adding links:
(client, sw0) (server1, sw0) (server2, sw0)
*** Configuring hosts
client server1 server2
*** Starting controller
c0
*** Starting 1 switches
sw0 ...
*** setting default gateway of host server1
server1 192.168.2.1
*** setting default gateway of host server2
server2 192.168.3.1
*** setting default gateway of host client
client 10.0.1.1
*** Starting SimpleHTTPServer on host server1
*** Starting SimpleHTTPServer on host server2
*** Starting CLI:
mininet>
```

3、现在你已经启动了模拟的网络拓扑，我们为你准备了一个已实现静态路由功能的路由器的可执行文件solution，供你测试网络拓扑的连通性，并以此来实现你的方案。运行以下命令，

```
cd ~/workspace/lab2/
./solution
```

你将看到以下输出：

```
vboxuser@ubuntu20:~/workspace/lab2$ ./solution
Using VNS sr stub code revised 2009-10-14 (rev 0.20)
Loading routing table from server, clear local routing table.
Loading routing table
---------------------------------------------
Destination Gateway     Mask    Iface
10.0.1.100      10.0.1.100  255.255.255.255 eth3
192.168.2.2     192.168.2.2 255.255.255.255 eth1
192.168.3.3     192.168.3.3 255.255.255.255 eth2
---------------------------------------------
Client vboxuser connecting to Server localhost:8888
Requesting topology 0
successfully authenticated as vboxuser
Loading routing table from server, clear local routing table.
Loading routing table
---------------------------------------------
Destination Gateway     Mask    Iface
10.0.1.100      10.0.1.100  255.255.255.255 eth3
192.168.2.2     192.168.2.2 255.255.255.255 eth1
192.168.3.3     192.168.3.3 255.255.255.255 eth2
---------------------------------------------
Router interfaces:
eth3    HWaddr4e:6c:b7:eb:69:08
    inet addr 10.0.1.1
eth2    HWaddr3e:3d:ef:f2:a2:79
    inet addr 192.168.3.1
eth1    HWaddr36:49:5c:75:db:74
    inet addr 192.168.2.1
 <-- Ready to process packets -->
```

4、现在回到运行Mininet的终端，执行ping命令，以测试网络的连通性。如果你需要在Mininet的某个主机上发出命令，那么需要你首先键入主机名，然后输入对应的命令。下图展示了一个在client上执行ping命令的例子。

```
mininet> client ping -c 3 server1
PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
64 bytes from 192.168.2.2: icmp_seq=1 ttl=63 time=348 ms
64 bytes from 192.168.2.2: icmp_seq=2 ttl=63 time=64.3 ms
64 bytes from 192.168.2.2: icmp_seq=3 ttl=63 time=94.5 ms

--- 192.168.2.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 64.259/169.074/348.465/127.447 ms
mininet>
```

你也可以使用traceroute命令查看client到server1之间的路由

```
mininet> client traceroute -n 192.168.2.2
traceroute to 192.168.2.2 (192.168.2.2), 30 hops max, 60 byte packets
 1  10.0.1.1  49.895 ms  55.864 ms  55.886 ms
 2  192.168.2.2  97.586 ms  97.628 ms  105.258 ms
mininet>
```

最后，请你使用wget命令发出HTTP请求，测试server1和server2上的Web服务器是否工作

```
mininet> client wget http://192.168.2.2
--2023-10-24 16:39:53--  http://192.168.2.2/
Connecting to 192.168.2.2:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 161 [text/html]
Saving to: ‘index.html.11’

index.html.11       100%[===================>]     161  --.-KB/s    in
 0s

2023-10-24 16:39:53 (15.3 MB/s) - ‘index.html.11’ saved [161/161]
```

5、在执行了上述流程后，相信你已经了解完成本任务所需的主要流程，现在你的任务是实现路由器中的主要逻辑，并将其编译后运行于实验拓扑中，使其具有与solution相同的静态转发的功能。

4.2.4 代码概览

本任务代码在目录 `/home/vboxuser/workspace/lab2/router` 下，你需要执行以下命令以编译你的代码：

```
cd ~/workspace/lab2/router
sudo make
cp sr ../sr
cd ..
./sr
```

在本任务中，你需要补全sr_router.c中空缺的代码，空缺的代码块以如下形式呈现：

```
/* Lab4-Task2 TODO */
需要你实现的代码
/* End TODO */
```

为了便于你理解代码，我们对sr_router.c中主要的方法做简要介绍：

```
void sr_handlepacket(struct sr_instance* sr, uint8_t * packet, unsigne
d int len, char* interface)
```

路由器每次收到数据包之后都会调用 `sr_handlepacket` 方法，sr参数为指向路由器实例的指针，packet参数指向该数据包的缓冲区，interface参数指接收该数据包的路由器接口名称。

```
void sr_handle_arp(struct sr_instance* sr, uint8_t * buf, unsigned int
 len, char* interface)
```

路由器调用 `sr_handle_arp` 方法以处理arp包，sr参数为指向路由器实例的指针，buf指向该数据包的缓冲区，interface参数指接收该数据包的路由器接口名称。

```
void sr_handle_ip(struct sr_instance* sr, uint8_t * buf, unsigned int
len,char* interface)
```

路由器调用 `sr_handle_ip` 方法处理ip包，其中的参数的含义与 `sr_handle_arp` 方法一致。

```
struct sr_rt *prefix_match(struct sr_instance * sr, uint32_t addr)
```

在 `prefix_match` 方法中实现了对一个IP地址做最长前缀匹配的过程，addr参数为传入的目的IP地址。

4.2.5 任务涉及的网络协议

IP协议

在本任务中，在对 IP 数据报进行操作之前，应验证其校验和，随后在路由表中查找目标 IP 地址的最长前缀匹配。如果路由器确定应转发数据报，则应正确减少数据包头部的 TTL 字段，并在将其转发到下一跳之前重新计算更新校验和。IP协议详情可查看RFC 791： Internet Protocol

地址解析协议ARP

在本任务中需要 ARP 来确定与路由表中存储的下一跳 IP 地址相对应的下一跳 MAC 地址。如果无法生成 ARP 请求和处理 ARP 回复，路由器将无法确认发送的以太网帧的目标 MAC 地址。对于 ARP 请求，只有当其目标 IP 地址是路由器的 IP 地址之一时，该路由器才应发送 ARP 回复。在收到 ARP 回复时，如果目标 IP 地址是路由器的 IP 地址之一，则应缓存该条目。请注意，ARP请求将发送到广播 MAC 地址 (ff-ff-ff-ff-ff-ff)。ARP 回复直接发送到请求者的 MAC 地址。ARP协议详情可查看RFC 826: An Ethernet Address Resolution Protocol

网际控制报文协议ICMP

在本任务中，路由器仅响应ICMP回送请求报文，并在以下情况中，发送对应的ICMP消息给源主机：

- 回送回答（type=0）：用以响应发送到路由器某一接口的回送请求。

- 终点不可达（type=3，code=0）：目标IP的路由不存在时，则通知源主机。
- 时间超过（type=11，code=0）： 如果由于 TTL 字段为 0，IP 数据包在处理过程中被丢弃，则通知源主机。 ICMP协议详情可查看RFC 792: Internet Control Message Protocol

4.2.6 测试样例

如果你的代码工作正常，则以下所有操作都应该有效：

- 从客户端 ping 到任何路由器端口。
- 从客户端 ping 到任何应用程序服务器（server1，server2）。
- 使用traceroute打印从客户端到任何应用程序服务器的路由。
- 使用HTTP从应用程序服务器之一下载文件。

我们为你准备了6个测试样例，在实现完你的方案后，你可以根据这些测试样例评测代码的正确性。当你提交你的实验代码后，我们也会根据这些测试样例来评测你的代码。

测试样例1：使用client对web服务器进行ping操作，运行如下指令，且需看到如下运行结果：

```
mininet> client ping -c 3 192.168.2.2
PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
64 bytes from 192.168.2.2: icmp_seq=1 ttl=63 time=222 ms
64 bytes from 192.168.2.2: icmp_seq=2 ttl=63 time=60.0 ms
64 bytes from 192.168.2.2: icmp_seq=3 ttl=63 time=47.2 ms

--- 192.168.2.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 47.246/109.862/222.335/79.700 ms
```

测试样例2：使用client对server1进行traceroute操作，运行如下指令，且需看到如下运行结果：

```
mininet> client traceroute 192.168.2.2
traceroute to 192.168.2.2 (192.168.2.2), 30 hops max, 60 byte packets
1 10.0.1.1 (10.0.1.1) 10.491 ms 11.328 ms 17.253 ms
2 192.168.2.2 (192.168.2.2) 65.967 ms 66.003 ms 69.908 ms
```

测试样例3：使用client对交换机端口进行ping操作，运行如下指令，且需看到如下运行结果：

```
mininet> client traceroute 192.168.3.1
traceroute to 192.168.3.1 (192.168.3.1), 30 hops max, 60 byte packets
1 10.0.1.1 (10.0.1.1) 47.876 ms 50.874 ms 53.287 ms
```

测试样例4：使用server1对server2进行ping操作，运行如下指令，且需看到如下运行结果：

```
mininet> server1 ping -c 3 server2
PING 192.168.3.3 (192.168.3.3) 56(84) bytes of data.
64 bytes from 192.168.3.3: icmp_seq=1 ttl=63 time=216 ms
64 bytes from 192.168.3.3: icmp_seq=2 ttl=63 time=52.0 ms
64 bytes from 192.168.3.3: icmp_seq=3 ttl=63 time=86.0 ms

--- 192.168.3.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2006ms
rtt min/avg/max/mdev = 51.952/118.064/216.251/70.805 ms
```

测试样例5：使用server1对server2进行traceroute操作，运行如下指令，且需看到如下运行结果：

```
mininet> server1 traceroute 192.168.3.3
traceroute to 192.168.3.3 (192.168.3.3), 30 hops max, 60 byte packets
1 192.168.2.1 (192.168.2.1) 41.597 ms 47.140 ms 47.161 ms
2 192.168.3.3 (192.168.3.3) 89.052 ms 93.094 ms 93.505 ms
```

测试样例6：使用client对server1进行wget操作，运行如下指令，且需要看到如下运行结果：

```
mininet> client wget http://192.168.3.3
--2023-10-27 19:32:26--  http://192.168.3.3/
Connecting to 192.168.3.3:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 161 [text/html]
Saving to: ‘index.html.2’

index.html.2        100%[===================>]     161  --.-KB/s    in
 0s

2023-10-27 19:32:26 (19.2 MB/s) - ‘index.html.2’ saved [161/161]
```

4.3 任务三：动态路由

在本任务中，你将在任务二静态路由的基础上，为路由器添加动态路由功能。同任务二一样，任务三的网络拓扑与代码框架均已给出。

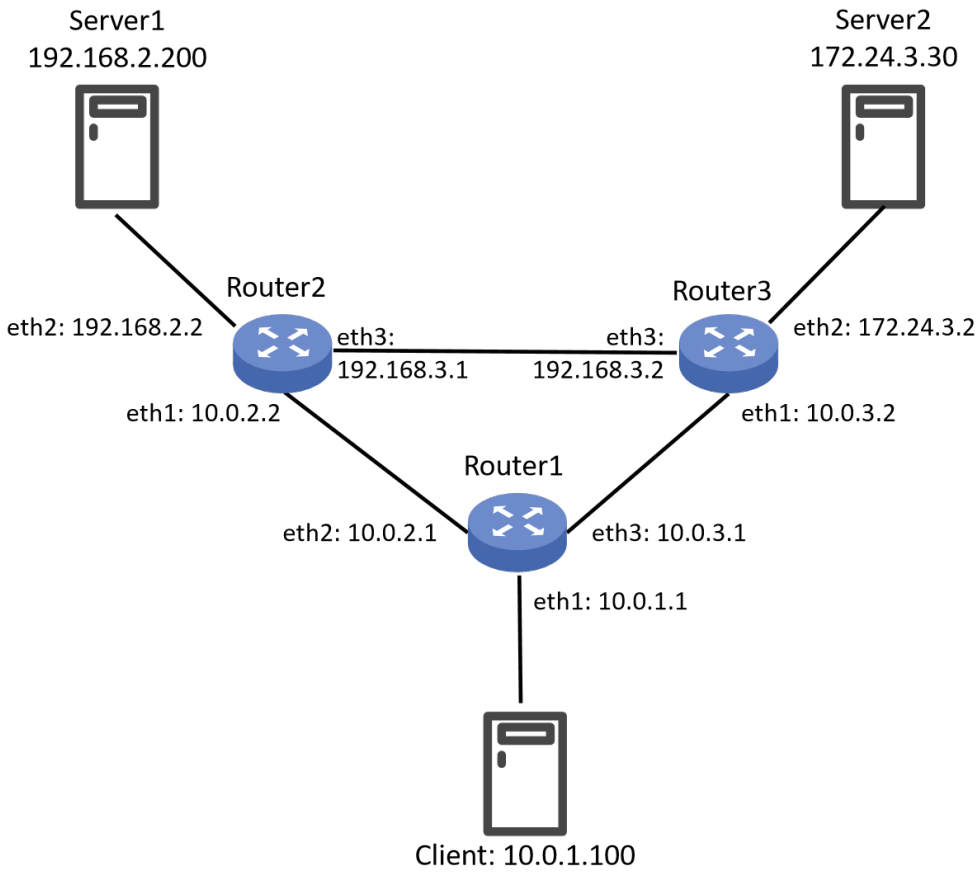
4.3.1 任务要求

请基于给定的拓扑及代码框架

1. 将动态路由代码补充完整，实现RIP路由协议。
2. 阅读代码，画出动态路由的代码流程图。

4.3.2 实验拓扑

任务三网络拓扑如下，其包含一台客户端（client）主机，三台需要你实现RIP协议的路由器（Router1, Router2, 和Router3），两台服务器（分别是server1和server2）主机，三台主机都连接于路由器，具体的IP和端口设置如下图所示。



4.3.3 实验流程

1、进入虚拟机并定位到具体的实验代码目录：

```
/home/vboxuser/workspace/lab3
```

2、进入对应的实验目录，并运行以下命令以配置运行环境、启动Mininet模拟器和POX控制器（POX控制器和Mininet需要分别运行在两个终端中）：

```
cd ~/workspace/lab3/
./config.sh
./run_pox.sh
./run_mininet.sh
```

执行 `./config.sh` ，你将能看到以下输出：

```
vboxuser@ubuntu20:~/workspace/lab3$ ./config.sh
running develop
running egg_info
writing requirements to pwospf.egg-info/requires.txt
writing pwospf.egg-info/PKG-INFO
writing top-level names to pwospf.egg-info/top_level.txt
writing dependency_links to pwospf.egg-info/dependency_links.txt
reading manifest file 'pwospf.egg-info/SOURCES.txt'
writing manifest file 'pwospf.egg-info/SOURCES.txt'
running build_ext
Creating /usr/local/lib/python2.7/dist-packages/pwospf.egg-link (link
to .)
pwospf 0.0.0 is already the active version in easy-install.pth

Installed /home/vboxuser/workspace/lab3/pox_module
省略后续输出
```

运行POX控制器时，你应该能看到如下输出：

```
vboxuser@ubuntu20:~/workspace/lab3$ ./run_pox.sh
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
server1 192.168.2.200
server2 172.24.3.30
client 10.0.1.100
router1-eth1 10.0.1.1
router1-eth2 10.0.2.1
router1-eth3 10.0.3.1
router2-eth1 10.0.2.2
router2-eth2 192.168.2.2
router2-eth3 192.168.3.1
router3-eth1 10.0.3.2
router3-eth2 172.24.3.2
router3-eth3 192.168.3.2
INFO:.home.vboxuser.workspace.lab3.pox_module.pwospf.srhandler:created
 server
SRServerListener listening on 8888
INFO:core:POX 0.5.0 (eel) is up.
```

运行Mininet模拟器时，你应该能看到如下输出：

```
vboxuser@ubuntu20:~/workspace/lab3$ ./run_mininet.sh
[sudo] password for vboxuser:
*** Shutting down stale SimpleHTTPServers
*** Shutting down stale webservers
server1 192.168.2.200
server2 172.24.3.30
client 10.0.1.100
router1-eth1 10.0.1.1
router1-eth2 10.0.2.1
router1-eth3 10.0.3.1
router2-eth1 10.0.2.2
router2-eth2 192.168.2.2
router2-eth3 192.168.3.1
router3-eth1 10.0.3.2
router3-eth2 172.24.3.2
router3-eth3 192.168.3.2
*** Successfully loaded ip settings for hosts
 {'server1': '192.168.2.200', 'server2': '172.24.3.30', 'router3-eth3':
*** Creating network
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
client server1 server2
*** Adding switches:
router1 router2 router3
*** Adding links:
(client, router1) (router2, router1) (router3, router1) (router3, rout
er2) (server1, router2) (server2, router3)
*** Configuring hosts
client server1 server2
*** Starting controller
c0
*** Starting 3 switches
router1 router2 router3 ...
*** setting default gateway of host server1
server1 192.168.2.2
*** setting default gateway of host server2
server2 172.24.3.2
*** setting default gateway of host client
client 10.0.1.1
*** Starting SimpleHTTPServer on host server1
*** Starting SimpleHTTPServer on host server2
*** Starting CLI:
```

3、现在已启动了模拟的网络拓扑，本任务提供了一个已实现RIP的路由器可执行文件solution

供参考，请在三个终端中分别执行下述的三个的三个命令，以分别运行三个路由器。

```
./solution -t 300 -s 127.0.0.1 -p 8888 -v router1
```

```
./solution -t 300 -s 127.0.0.1 -p 8888 -v router2
```

```
./solution -t 300 -s 127.0.0.1 -p 8888 -v router3
```

对任一路由器，你将看到下列内容，打印的内容中显示了路由表各端口和转发表的情况

```
vboxuser@ubuntu20:~/workspace/lab3$ ./solution -t 300 -s 127.0.0.1 -p
8888 -v router1
Using VNS sr stub code revised 2009-10-14 (rev 0.20)
Client vboxuser connecting to Server 127.0.0.1:8888
Requesting topology 300
successfully authenticated as vboxuser
Router interfaces:
eth3    HWaddr86:5f:a9:3d:c9:85
    inet addr 10.0.3.1
    inet mask 255.255.255.252
eth2    HWaddra2:b2:40:bf:95:c6
    inet addr 10.0.2.1
    inet mask 255.255.255.252
eth1    HWaddrfa:4e:39:6b:cd:00
    inet addr 10.0.1.1
    inet mask 255.255.255.0
  <---------- Router Table ---------->
Destination Gateway     Mask        Iface   Metric  Update_Time
10.0.3.0    0.0.0.0 255.255.255.252 eth3    0   14:44:23
10.0.2.0    0.0.0.0 255.255.255.252 eth2    0   14:44:23
10.0.1.0    0.0.0.0 255.255.255.0   eth1    0   14:44:23
 <-- Ready to process packets -->
```

4、在提供的参考实现中(solution)，当有两个及以上路由器运行时，路由器之间将每隔五秒钟自动交换路由信息。每五秒钟，你将看到每个路由器中更新后的路由信息，也即路由器中路由表的内容（下为router1的路由表内容）。

```
  <---------- Router Table ---------->
Destination Gateway     Mask        Iface   Metric  Update_Time
10.0.3.0    0.0.0.0 255.255.255.252 eth3    0   14:44:53
10.0.2.0    0.0.0.0 255.255.255.252 eth2    0   14:44:53
10.0.1.0    0.0.0.0 255.255.255.0   eth1    0   14:44:53
192.168.3.0 10.0.2.2    255.255.255.252 eth2    1   14:44:52
192.168.2.0 10.0.2.2    255.255.255.0   eth2    1   14:44:52
172.24.3.0  10.0.3.2    255.255.255.0   eth3    1   14:44:50
```

5、当路由表收敛后，你可以在Mininet中使用traceroute命令查看client到server1的路由信息，你将能看到如下输出：

```
mininet> client traceroute 192.168.2.200
traceroute to 192.168.2.200 (192.168.2.200), 30 hops max, 60 byte pack
ets
 1  10.0.1.1 (10.0.1.1)  63.712 ms  63.589 ms  63.627 ms
 2  10.0.2.2 (10.0.2.2)  901.800 ms  831.979 ms  901.836 ms
 3  192.168.2.200 (192.168.2.200)  1019.533 ms  1019.655 ms  1019.596 ms
```

6、现在我们手动断开router1和router2之间的链路，再次查看路由表的收敛情况。在Mininet中使用下述命令。

```
mininet> link router1 router2 down
```

你将能观察到路由器的路由表在几分钟后收敛，并且router1的路由表没有了到router2的路由

```
  <---------- Router Table ---------->
Destination Gateway     Mask         Iface   Metric  Update_Time
10.0.3.0    0.0.0.0 255.255.255.252 eth3    0   14:46:58
10.0.1.0    0.0.0.0 255.255.255.0   eth1    0   14:46:58
192.168.3.0 10.0.3.2    255.255.255.252 eth3    1   14:46:55
192.168.2.0 10.0.3.2    255.255.255.0   eth3    2   14:46:55
172.24.3.0  10.0.3.2    255.255.255.0   eth3    1   14:46:55
```

7、再次在Mininet中使用traceroute，此时你将发现client到server1的路由信息已经发生了更新。

```
mininet> client traceroute server1
traceroute to 192.168.2.200 (192.168.2.200), 30 hops max, 60 byte pack
ets
 1  10.0.1.1 (10.0.1.1)  74.457 ms  74.413 ms  74.404 ms
 2  10.0.3.2 (10.0.3.2)  147.585 ms  150.291 ms  213.970 ms
 3  192.168.3.1 (192.168.3.1)  271.000 ms  326.139 ms  394.015 ms
 4  192.168.2.200 (192.168.2.200)  394.040 ms  394.027 ms  394.019 ms
```

8、若恢复在第6步中断开的链路后，再次使用traceroute查看client到server1的路由信息，可发现其又恢复到步骤5中的状态。

```
mininet> link router1 router2 up
mininet> client traceroute server1
traceroute to 192.168.2.200 (192.168.2.200), 30 hops max, 60 byte pack
ets
 1  10.0.1.1 (10.0.1.1)  91.427 ms  91.435 ms  36.478 ms
 2  10.0.2.2 (10.0.2.2)  302.904 ms  351.201 ms  351.218 ms
 3  192.168.2.200 (192.168.2.200)  399.408 ms  399.498 ms  442.537 ms
```

4.3.4 任务代码概览

本任务涉及的代码在目录 `/home/vboxuser/workspace/lab3/router` 下，在开始本任务前，请在确保任务二代码可正确运行后，将在任务二中补全后的sr_router.c文件拷贝到任务三/lab3/router目录下。 你可以执行以下命令以编译你的代码：

```
cd ~/workspace/lab3/router
sudo make
cp sr ../sr
cd ..
```

你需要在三个终端中分别执行下述的三个命令，以分别运行三个路由器。

```
./sr -t 300 -s 127.0.0.1 -p 8888 -v router1
```

```
./sr -t 300 -s 127.0.0.1 -p 8888 -v router2
```

```
./sr -t 300 -s 127.0.0.1 -p 8888 -v router3
```

本任务中需要补全的代码均在sr_rt.c文件中，空缺的代码块将以如下形式呈现：

```
/* Lab4-Task3 TODO */
需要你实现的代码
/* End TODO */
```

为了便于你理解本任务代码，现对sr_rt.c中主要的方法做简要介绍：

```
int sr_load_rt(struct sr_instance* sr,const char* filename)
```

路由器通过这个函数从现有的文件中读取路由表，sr参数为指向路由器实例的指针，filename参数为指向带有路由表的文件路径。

```
void sr_add_rt_entry(struct sr_instance* sr, struct in_addr dest,struc
t in_addr gw, struct in_addr mask, uint32_t metric, char* if_name)
```

路由器使用这个函数添加路由表的路由条目，sr参数为指向路由器实例的指针，dest参数为路由条目的目的IP地址，gw参数为路由条目的网关，mask参数为路由条目的掩码，metric参数为到该路由地址所需要的跳数，if_name参数表示转发到该路由所经过的端口名。

```
void sr_print_routing_table(struct sr_instance* sr)
```

路由器通过这个函数打印路由表，可以通过该函数观察路由表的收敛情况。

```
void sr_print_routing_entry(struct sr_rt* entry)
```

上一个函数通过调用这个函数来打印路由表中的具体条目。entry参数为执行某个条目的指针。

```
void *sr_rip_timeout(void *sr_ptr)
```

路由器通过调用这个函数，定时查看路由表中是否存在过期条目，是否存在有端口离线或端口上线的情况发生，如果存在这些情况，则给邻近的交换机发布RIP路由更新消息，更新临近交换机的路由表。

```
void send_rip_request(struct sr_instance *sr)
```

路由器通过这个函数发送RIP的请求，调用这个函数时，会一次性向所有路由器端口发送RIP请求包。

```
void send_rip_response(struct sr_instance *sr)
```

路由器通过这个函数发送RIP回复，调用这个函数时，会一次性向所有的路由发送自身的路由表。

```
void update_route_table(struct sr_instance *sr, uint8_t *packet, unsig
ned int len, char *interface)
```

路由器通过该函数接受其他路由的回复，收到其他路由回复后，与自身的路由表做比较，如果路由表发生改变，则路由器自身调用路由回复函数，向相邻的路由器发布新的路由表。

4.3.5 涉及的网络协议

本任务使用RIPv2，请阅读RFC2453 RIP Version 2了解更多RIPv2细节。请注意，RIPv2使用多播将路由表发送给邻近路由器，为了简化代码实现，本任务中使用广播发送路由表。

4.3.6 测试样例

如果你的代码工作正常，则以下所有操作都应该有效：

- 在给定的网络拓扑中能正确计算路由表。
- 当网络拓扑发生变化时，网络中路由能正确收敛。

我们为你准备了15个测试样例，在实现完你的方案后，你可以根据这些测试样例评测代码的正确性。当你提交你的实验代码后，我们也会根据这些测试样例来评测你的代码。

你可以在Mininet中执行以下命令以验证测试用例：

```
mininet> source testcases.sh
```

如果你通过了所有测试用例，你将看到如下输出：

```
mininet> source testcases.sh
Testcase1:  Pass
nohup: appending output to 'nohup.out'
Testcase2:  Pass
nohup: appending output to 'nohup.out'
Testcase3:  Pass
Info: router1-eth2 down
Info: Sleep 60 seconds, wait for routing table converging
Testcase4:  Pass
Testcase5:  Pass
Testcase6:  Pass
Info: router2-eth3 down
Info: Sleep 60 seconds, wait for routing table converging
Testcase7:  Pass
Testcase8:  Pass
Testcase9:  Pass
Info: router2-eth3 up
Info: router1-eth2 up
Info: Sleep 60 seconds, wait for routing table converging
Testcase10:  Pass
Testcase11:  Pass
Testcase12:  Pass
Info: router1-eth1 down
Info: Sleep 60 seconds, wait for routing table converging
Testcase13:  Pass
Testcase14:  Pass
Info: router1-eth1 up
Info: Sleep 60 seconds, wait for routing table converging
Testcase15:  Pass
```

测试用例的具体设置如下：

测试样例1： Client Ping往拓扑中所有的IP地址，并traceroute到 server1。

测试样例2： Server1 Ping往拓扑中所有的IP地址，并traceroute到server2。

测试样例3： Server2 Ping往拓扑中所有的IP地址，并traceroute到client。

进行到这一步之后，断开router1和router2之间的链路，然后测试以下测试样例

测试样例4： Client Ping往拓扑中所有的IP地址，并traceroute到server1。

测试样例5： Server1 Ping往拓扑中所有的IP地址，并traceroute到server2。

测试样例6： Server2 Ping往拓扑中所有的IP地址并traceroute到client。

进行到这一步后，断开router2和router3之间的链路，测试以下测试样例

测试样例7： Client Ping往拓扑中所有的IP地址，并traceroute到server2。

测试样例8： Server1 Ping往拓扑中所有的IP地址。

测试样例9： Server2 Ping往拓扑中所有的IP地址，并traceroute 到client。

然后恢复我们刚才断开的2个链路，测试以下测试样例

测试样例10： 与测试用例1相同。

测试样例11： 与测试用例2相同。

测试样例12： 与测试用例3相同。

然后阻塞 router1 和 client 之间的链路，测试以下测试样例

测试样例13： Server1 Ping往拓扑中所有的IP地址，并traceroute到 server2。

测试样例14： Server2 Ping往拓扑中所有的IP地址，并traceroute到router1-eth3。

最后，接收router1和client之间的链路，测试以下测试样例

测试样例15： 与测试用例1相同。

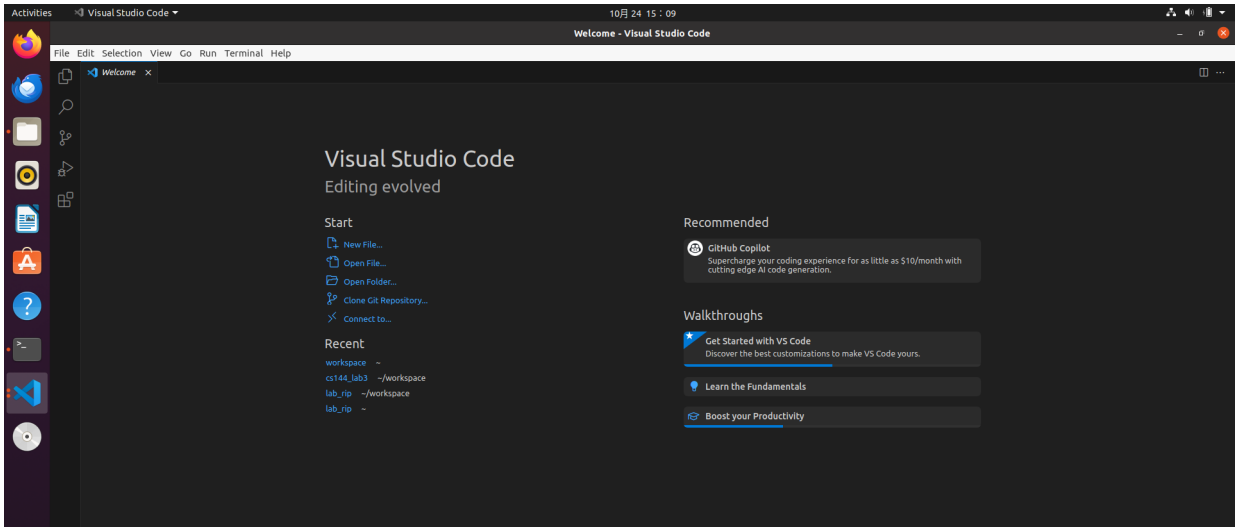
五、相关工具与文档

5.1 开发工具

我们在虚拟机中为你安装了vscode，以便于你编写实验代码，你也可以安装自己的喜好选择合适的IDE。若需要使用vscode，请你执行以下命令：

```
code
```

你将能看到IDE启动，并在其中看到我们预先设置的实验代码。



5.2 相关文档

为了便于同学们课后学习，在本指导书中为同学们提供了实验中涉及到的工具和开源软件的官网和代码。

- Mininet
- Mininet Github
- Pox Github
- Wireshark