路由器开发实验指导

第一部分搭建路由器

利用现有软件搭建路由器。路由协议采用 Quagga，转发引擎利用 Linux 内核。

1 实验目的

1）了解路由器的结构

2）熟悉软件路由器的搭建方法

3）熟悉路由器静态路由配置

4）熟悉路由器协议 RIP 的配置

2 实验内容

2.1 实验一：软件下载、安装及配置

（1）实验环境

如图 1.1 所示，所有学生机通过网络与测试服务器连接。

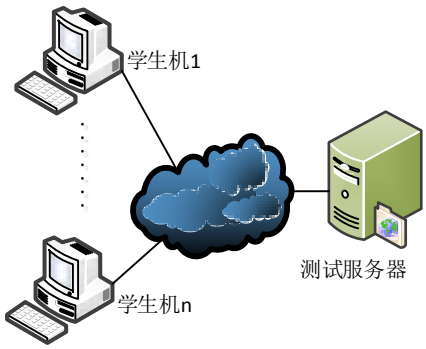

图 1.1 路由开发环境

（2）实验操作

从测试服务器（或互联网）下载 Quagga 软件，编译安装软件，并进行配置，从而完成软件路由器的搭建，可以进行路由器操作（如显示接口状态、配置接口地址等）。

在 CentOS-6.5 系统 64 位机上安装 quagga-0.99.21 过程（**以下安装为完整过程，若系统中已存在相关软件模块，可省略相关步骤**）：

1）安装 g++：

yum install gcc　yum install gcc-c++

2）安装 gawk：

yuminstall gawk

3）安装 ncurses：

yum install ncurses-libs

yum install ncurses-devel

4）安装 libreadline:

第一种方法：yum install readline-devel

如果第一种方法在编译 quagga-0.99.21 时出现 vtysh 这个目录下不能 make 过的情况，请使用第二种方法安装 readline。

第二种方法：

下载 readline-6.2.tar.gz

wget -c ftp://ftp.gnu.org/gnu/readline/readline-6.2.tar.gz

解压 readline-6.2.tar.gz

tarxzf readline-6.2.tar.gz

编译安装

进入解压出来的目录中

./configure

make&& make install

ldconfig

5）安装 quagga-0.99.21

1.解压压缩包：

tarxzf quagga-0.99.21_20131218.tar.gz

2.编译安装

./configure--enable-vtysh --enable-zebra --enable-ripd--disable-doc --disable-babeld --enable-isisd=no --disable-bgpd --disable-ospfd --disable-ospf6d --disable-ospfclient--enable-user=root --enable-group=root -enable-vty-group=root

make&& make install

加载库文件路径

ldconfig

创建 log 目录

mkdir /var/log/quagga/

编辑配置文件

cp/usr/local/etc/zebra.conf.sample　/usr/local/etc/zebra.conf

cp/usr/local/etc/ripd.conf.sample/usr/local/etc/ripd.conf

进入/usr/local/etc/zebra.conf 文件：

vim /usr/local/etc/zebra.conf 进行修改

将最后一行改成 log file /var/log/quagga/zebra.log

6）启动程序

zebra 启动：

zebra　-d　启动 zebra

zebrad　-d 启动后用 ps－ef | grep zebra 能看到已经启动的 zebra 程序

vtysh 启动：

zebra 启动后输入 vtysh 启动 vtysh

2.2 实验二：转发功能测试

（1）实验环境

如图 1.2 所示，学生机的两个网卡分别通过网络与测试服务器连接。

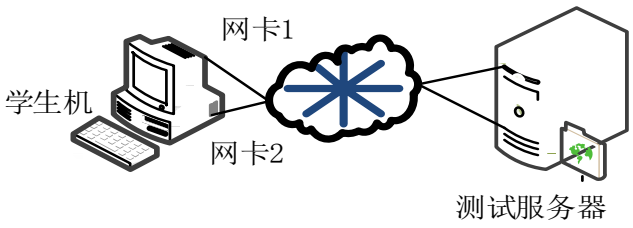

图 1.2 转发功能环境

（2）实验操作

学生机的网卡 1 与测试服务器的一个网卡相连，且配置同一网段 IP 地址；学生机的网卡 2 与测试服务器另一个网卡相连，且配置同一网段 IP 地址。学生机两个网卡的 IP 地址不在同一网段。运行 zebra，在学生机上配置静态路由，下一跳为

网卡 2 直连的网络接口 IP 地址，目的 IP 为自己指定的另一网段 IP 地址，测试服务器向学生机发送目的 IP 地址 ping 包，在测试服务器和网卡 2 相连的网络接口使用抓包工具，若学生机转发成功，可以抓到测试服务器发出的 ping 包。

2.3 实验三：路由协议 RIP 配置

（1）实验环境

如图 1.3 所示，学生机通过网络与测试服务器连接；学生机和测试服务器上都运行路由协议 RIP。

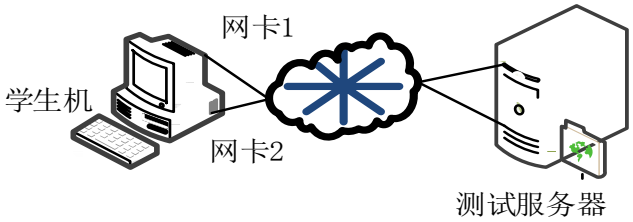

图 1.3RIP 配置环境

（2）实验操作

学生机上先执行 Ctrl+D，跳出到 Linux 界面，执行 ripd-d，后台运行 RIP。在测试服务器上也配置上 RIP。**要求：**学生机能够与测试服务器建立邻居关系，并且能够相互学习到对方路由，在测试服务器上配置一条静态路由，学生机可以通过 RIP 学到这条路由。

第二部分开发转发引擎

路由协议软件采用 Quagga，转发引擎需要自己开发。

1 实验目的

1）了解转发表的结构

2）了解路由查找基本原理

3）了解 IP 数据包的数据格式

4）熟悉报文转发基本流程

2 实验流程图

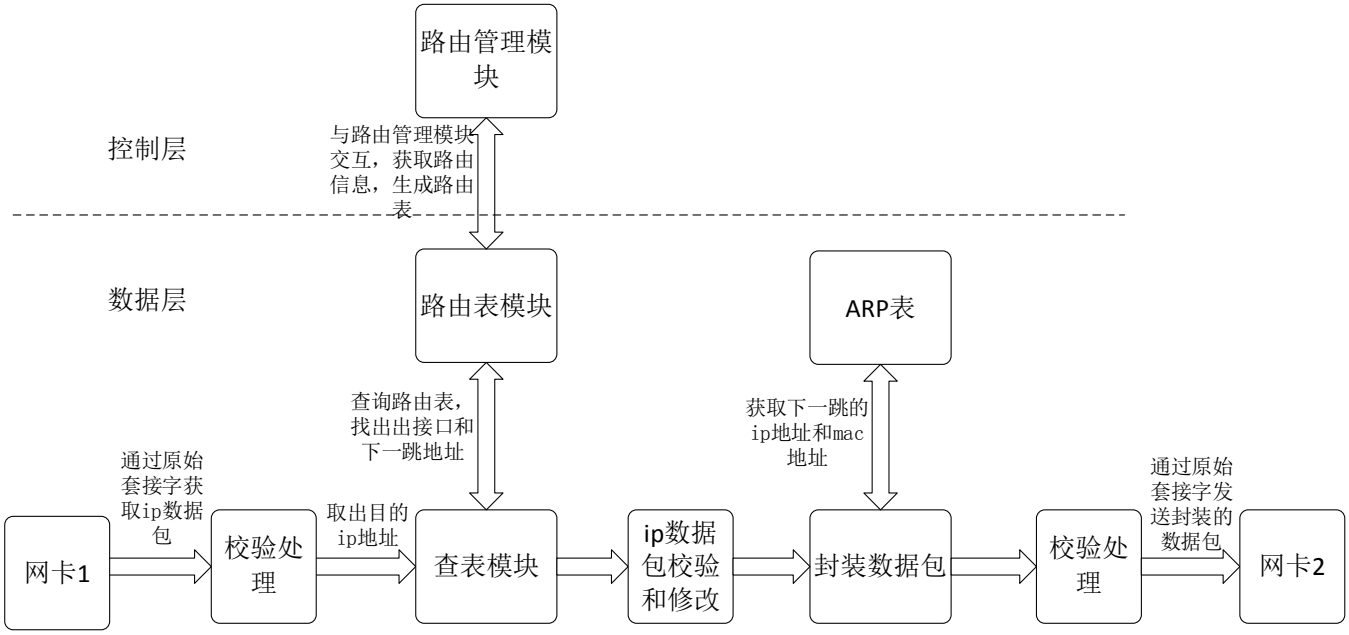

图 2.1 转发流程图

3 实验内容

3.1 实验一：构建全局路由表

（1）实验环境

在学生机上实现。

（2）实验操作

构建全局路由表，编写插入函数，通过链表（或其它数据结构）存储的方式将自定义的路由表项存储在路由表里。

插入函数：

intinsert_route(unsignedlongip4prefix,unsignedintprefixlen,char

*ifname,unsignedintifindex,unsigned long nexthopaddr)

参数 1 是目的地址，参数 2 是掩码长度，参数 3 是接口名，参数 4 是接口索引值，参数 5 是下一跳 IP 地址。

3.2 实验二：校验和计算

（1）实验环境

学生机上实现。

（2）实验操作

- 校验和验证函数

intcheck_sum(unsigned short *iphd,intlen,unsigned short checksum)

参数 1 是指向 IP 数据包头的指针 ip_recvpkt，参数 2 是 IP 数据包头长度，参数 3 是 IP 数据包的校验和

- 重新计算校验和函数

unsigned short count_check_sum(unsigned short *iphd)

参数是指向 IP 数据包头的指针 ip_recvpkt。

经过本机路由器转发的数据包，ttl 减 1，重新计算校验和，以便封装 IP 数据包头时使用。

3.3 实验三：编写查表函数和撤销路由函数

（1）实验环境

学生机上实现。

（2）实验操作

提取接收 IP 数据包的目的 IP 地址查找路由表，获取出接口和下一跳 IP 地址；撤销路由函数用来撤销失效的路由表项。

- 查表函数

intlookup_route(structin_addrdstaddr , structnextaddr *nexthopinfo)

参数 1 是目的 IP 地址 ip_recvpkt->ip_dst，参数 2 是存储下一跳和出接口信息的结构体。

- 撤销路由函数

intdelete_route(structin_addrdstaddr , unsigned intprefixlen)

参数 1 是目的 IP 地址，参数 2 是掩码长度。

3.4 实验四：接收静态路由信息并存储

（1）实验环境

学生机上实现。

（2）实验操作

创建 tcp socket 服务器端，调用线程来监听接收 quagga 传送的静态路由信息，再调用插入函数将其存储在路由表里，撤销静态路由时调用撤销路由函数删除在路由表的表项；

从 quagga 路由管理模块获取静态路由信息具体如下：

zebra_rib.c 文件里的 rib_install_kernel 函数下发路由信息到 Linux 内核。

rib_install_kernel 函数-->

kernel_add_ipv4 函数-->

netlink_route_multipath 函数

struct prefix 存放前缀信息，struct rib 存放下一跳信息。

可以在这里用 tcp socket 创建客户端，将路由信息发送给作为服务器端的路由转发处理程序。

3.5 实验五：重新封装数据包

（1）实验环境

学生机上实现。

（2）实验操作

- 填充以太网包头

将获取到的下一跳接口信息存储到存储接口信息的结构体 ifreq 里，通过 ioctl 获取出接口的 MAC 地址作为以太网包头的源 MAC 地址，目的 MAC 地址为下一跳 IP 地址对应的 MAC 地址，目的 MAC 地址可以通过 ARP 表项获取，据获取的信息封装以太网包头，以太网类型 eth_header->ether_type = htons(ETHERTYPE_IP)；

- 填充 IP 数据包头，进行校验；
- 将剩余的 IP 数据包信息填充到后面，通过 raw socket 将重新封装的数据包发送出去。

4 主要实现功能说明

（1）创建全局路由表链表，调用插入函数向路由表中插入表项

路由表结构体如下：

```
structroute_table
{
structroute_table*next;        /* Link list. */
structin_addr ip4prefix;/* ip prefix */
u_charprefixlen; /* ipprefixlen */
int priority; /*priority*/
structnexthop*nexthop; /* Nexthop structure */
};
```

下一跳信息结构体如下：

```
structnexthop
{
structnexthop*next;
char*ifname;
unsignedintifindex;/* Interface index. */
structin_addrnexthopaddr;/* Nexthop address */
}
```

（2）创建 tcp socket 服务器端，调用线程来监听接收 quagga 传送的静态路由信息，将其存储在路由表里。

quagga 路由管理模块获取静态路由信息如下：

zebra_rib.c 文件里的 netlink_route_multipath 函数下发路由信息到 Linux 内核。

struct prefix 存放前缀信息，struct rib 存放下一跳信息。

可以在这里用 tcp socket，将 quagga 路由管理模块作为客户端，将路由信息发送给作为服务器端的路由转发处理程序，收到路由消息后存储进路由表。

（3）编写撤销路由函数，以便撤销失效的路由

（4）通过 rawsocket 套接字在数据链路层捕获 IP 数据包

```
intrecvfd ;
ssize_trecvlen;
recvfd=socket(AF_PACKET, SOCK_RAW, htons(ETH_P_IP));
recvlen=recv(recvfd, skbuf, sizeof(skbuf), 0);
```

创建套接字，然后接收 IP 数据包，IP 数据包存储在 skbuf 里，structip * ip_recvpkt; 让 ip_recvpkt 指向 skbuf 中的 IP 数据包头；

（5）对 IP 数据包进行校验处理，校验和正确则继续，错误则丢弃该数据包

（6）取出 IP 数据包的目的 IP 地址，根据此信息查询路由表，获取路由表中相匹配的出接口信息和下一跳 IP 地址；

（7）通过查询 ARP 表得到下一跳的 MAC 地址

获取下一跳 MAC 地址函数:

intarpGet(char *ifname, char *ipStr)

参数 1 是下一跳接口名称，参数 2 是下一跳 IPv4 地址。

（8）将 IP 数据包包头的 TTL 减 1，再重新计算校验和

（9）重新封装数据包，通过 raw socket 从查询路由表获取的出接口经数据包发送出去

- 将获取到的下一跳接口信息存储到存储接口信息的结构体 ifreq 里，通过 ioctl 获取出接口的 MAC 地址作为以太网包头的源 MAC 地址，目的 MAC 地址为下一跳 IP 地址的 MAC 地址，根据获取的信息封装以太网包头，以太网类型 eth_header->ether_type = htons(ETHERTYPE_IP);
- 再填充 IP 数据包头，进行校验；
- 通过 raw socket 将重新封装的数据包发送出去。

第三部分开发 RIP 协议

RIP 路由协议采用 Quagga，需要开发 RIP 报文的发送和接收处理，以及路由计算。

1 实验目的

1）了解 RIP 协议原理

2）了解 RIP 协议报文交互过程

3）了解 RIP 协议报文处理过程

4）熟悉 RIP 协议报文的发送和接收

2 实验原理

RIP 是一种内部网关协议（IGP），是一种动态路由选择协议，用于自治系统（AS）内的路由信息的传递。RIP 协议基于距离向量算法，使用“跳数”（即 metric）来衡量到达目标地址的路由距离。在默认情况下，RIP 使用一种非常简单的度量制度：距离就是通往目的站点所需经过的链路数，取值为 1~15，数值 15 表示路径无限长。RIP 进程使用 UDP 的 520 端口来发送和接收 RIP 分组。RIP 分组分为两种：请求分组和响应分组。

启动 RIP 之后，在配置 network 之后，会向直连主机组播发送 RIP 请求报文，请求一份完整的路由表，然后开始接收 RIP 报文，直连主机在接收到 request 请求报文后，会查询自身的 RIP 路由表，获取 RIP 路由表信息，然后修改命令为 response，再组播发送回去，自身路由发送变化时，会通过更新路由组播发送给直连路由器。如图 3.1 所示。

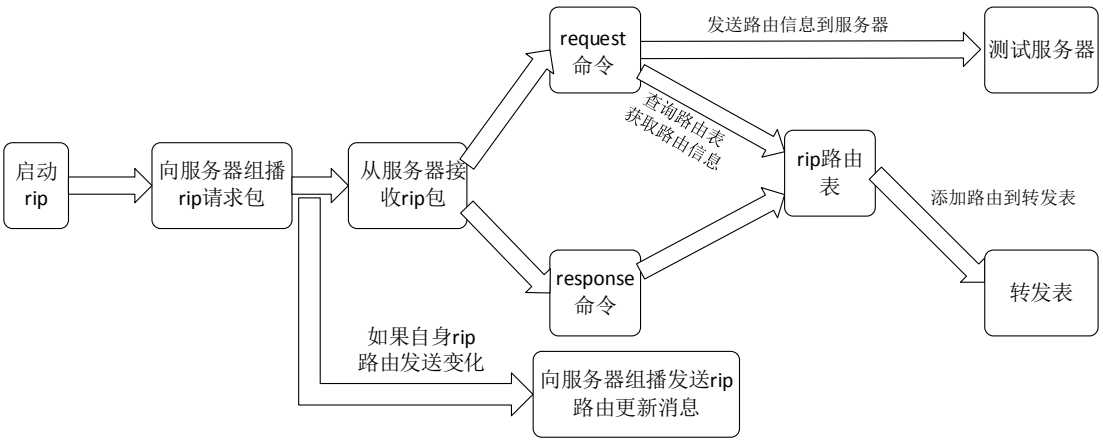

图 3.1 RIP 协议流程图

RIP 报文格式如下：报文头部是 4 个字节，包括 了“版本”字段 2 个字节和“命令”字段 2 个字节。路由表项是 16 个字节，其中地址族标识占 2 个字节，路由标记占 1 个字节，度量值占 1 个字节，子网掩码、下一跳、IP 地址各自占 4 个字节。一个完整的 RIP 报文可以有多个路由表项，但报文的总长度不能超过 udp 报文长度。详见表 3.1。

版本 version(2)	命令 command(2)	
地址族标识 AF(2)	路由标记 Tag(1)	度量值 metric(1)
子网掩码(4)		
下一跳(4)		
IP 地址(4)		

表 3.1 RIP 报文格式

version:RIP 报文版本，默认情况下为 2；

command:定义了报文类型(response 或者 request)；

AF:Address family identifier,发送请求报文时地址族默认为 0；

Tag:外部路由标记；

metric:路由跳数，请求完整的 RIP 报文时该值为 16；

子网掩码:用于区分子网；

Next Hop:下一跳地址；

IP 地址:目标地址 IP。

3 实验内容

3.1 实验一：RIP 请求报文的发送和接收以及 RIP 更新报文的发送

（1）实验环境

如图 3.2 所示，学生机通过网络与测试服务器连接；利用学生机上的网卡 1 和测试服务器相连接，测试服务器上运行路由协议 RIP。

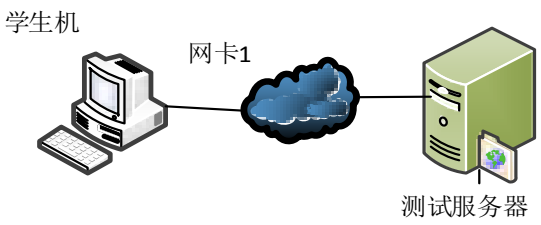

图 3.2RIP 请求报文开发环境

（2）实验操作

步骤 1:获取 RIP 代码，搭建环境

- 获取支持 RIP 的 quagga 代码，在 PC 机上构建运行和开发环境。
- 熟悉和了解 RIP 协议的原理，为开发 RIP 协议的模块做准备。

步骤 2:实现 RIP 发送处理模块

2.1:RIP 发送请求报文，实现 rip_request_interface_send 函数。

- 判断 RIP 协议版本是否为 RIPv2 并且支持组播；
- 填充请求完整的 RIP 报文内容，command 字段为 RIP_REQUEST；
- 调用 rip_send_packet 函数将报文发送给每一个直连网络。

2.2:RIP 发送更新报文，实现 rip_update_process 函数。

2.2.1 发送更新给每一个直连网络：

- 使用 ALL_LIST_ELEMENTS_RO 函数遍历每一个接口；
- 查看是否是 loopback 口，是的话则 continue；
- 获取 RIP 接口信息，当指定被动接口时，抑制响应；
- 使用 ALL_LIST_ELEMENTS 遍历每一个直连网络，使用 rip_update_interface 发送更新到每一个直连网络，其地址族需为 AF_INET。

2.2.2 发送更新给每一个邻居：

- 使用 route_top 和 route_next 来遍历每一个邻居；
- 使用 if_lookup_address 和 connected_lookup_address 来确保邻居是连接网络的；
- 设置目的地址和端口，使用 rip_output_process 发送更新报文给邻居。

步骤 3:实现 RIP 接收处理模块

3.1:RIP 接收数据处理，实现 rip_read 函数。

- 使用 THREAD_FD 函数获取套接字，使用 recvfrom 接收数据包；
- if_check_address 函数检查是否从自己接口发出，是则忽略该数据包；
- 根据 if_lookup_address 函数丢弃不正确接口的数据包；

- 根据 if_lookup_address 和 connected_lookup_address 找出 ipv4 地址；
- RIP 协议版本的检验；
- 请求报文或者响应报文处理。

3.2:RIP 请求报文处理，实现 rip_request_process 函数。

- 使用 if_is_loopback 函数，判断是否不响应 loopback 接口；
- 检查接口是否 enable；
- 当指定被动接口时，抑制响应；
- 更新 RIP 邻居 rip_peer_update；
- 如果是请求完整的 RIP 路由表，调用 rip_output_process 响应请求发送路由；
- 若不是请求完整的 RIP 路由表，则按照条目处理请求报文，调用 rip_send_packet 函数发送。

3.2 实验二：RIP 响应报文的接收

（1）实验环境

如图 3.3 所示，学生机通过网络与测试服务器连接；利用学生机上的网卡 1 和测试服务器相连接，测试服务器上运行路由协议 RIP。

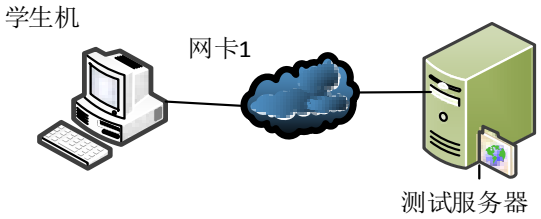

图 3.3 RIP 响应报文开发环境

（2）实验操作

步骤 1:获取 RIP 代码，搭建环境

- 获取支持 RIP 的 quagga 代码，在 PC 机上构建运行和开发环境。
- 熟悉和了解 RIP 协议的原理，为开发 RIP 协议的模块做准备。

步骤 2:实现 RIP 接收处理模块

参考实验三中第一个小实验步骤 3 中的 3.1 rip_read 函数。

步骤 3:实现 RIP 响应报文处理模块

3.1:RIP 响应报文处理，实现 rip_response_process 函数。

- 检查包是否是从 RIP 接口（520）接收的，不是则丢弃；
- 使用 if_lookup_address 函数检验数据报的 ipv4 源地址是否是直连的邻居；
- 使用 rip_peer_update 函数，更新 RIP 邻居；
- 获得路由条目，遍历路由条目，检验每一个条目：

检验过程：

- 地址族校验，RIP 仅支持 AF_INET；
- 使用 rip_destination_check 函数判断目的地址是否可靠；
- 使用 if_lookup_address 函数查看下一跳是否存在；
- 如果下一跳是组播地址,则忽略该条目;使用 if_lookup_address 查询下一跳地址是否存在，存在返回非空，使用 route_node_match_ipv4 在路由表中匹配下一跳地址,如果匹配到,若路由的类型为 ZEBRA_ROUTE_RIP 并且路由的 sub 类型为 RIP_ROUTE_RTE，则设置下一跳为源地址；若不是，下一跳设为 0；如果匹配不到，下一跳也设为 0。
- 使用 rip_rte_process 函数更新 RIP 路由表以及发送内核。

3.2:更新 RIP 路由表以及发送内核，实现 rip_rte_process 函数。

- 填充 prefix 结构体；

```
struct prefix_ipv4
{
u_charfamily;
u_charprefixlen;
structin_addrprefix __attribute__ ((aligned (8)));
};
```

- 使用 apply_mask_ipv4 函数确保掩码是实用的；
- 使用 rip_offset_list_apply_in 函数处理偏移列表；
- 如果偏移列表不修改 metric,使用接口的度量(metric)，如果度量值大于 16，设置度量值为 16；

- 设置下一跳指针，如果下一跳为空,设置为源地址,非空,设置为条目的下一跳；
- 使用 rip_nexthop_check 函数检查下一跳地址是否是自己，是直接返回；
- 使用 route_node_get 函数从路由表里获取前缀的索引；
- 检查路由表里是否已经有了路由：
  - **如果有路由：**
    - 将下一跳地址与数据报的路由器地址进行比较。如果该数据报来自与现有路由相同的路由器，则 rip_timeout_update 函数重新初始化超时时间；
    - 比较度量，如果数据报与现有路由来自同一路由器，并且新度量与旧度量不同；或者新度量低于旧度量，或者如果标签已经改变;或者存在具有较低管理距离的路由，或者更新实际 ROU 上的距离。做下列操作：
      - 根据数据报的路由，填写新的路由信息并调整下一跳的地址；
      - 如果旧的 metric 为无穷大，则新的 metric 不为无穷大,设置路类型为 ZEBRA_ROUTE_RIP，sub 类型为 RIP_ROUTE_RTE；使用 RIP_TIMER_OFF 函数关闭定时器；调整下一跳，使用 rip_zebra_ipv4_add 添加路由到内核;设置路由标志；
      - 如果旧的 metric 不为无穷大,使用 rip_zebra_ipv4_delete 函数从内核删除原来的路由；调整下一跳,使用 rip_zebra_ipv4_add 添加路由到内核并更改路由标志；
      - 调用 rip_event(RIP_TRIGGERED_UPDATE, 0);通知进程触发更新；
      - 如果新度量为无穷大，如果旧的 metric 不为无穷大，使用 RIP_TIMER_ON 和 RIP_TIMER_OFF 函数将垃圾

收集计时器设置为 120s,使用 rip_zebra_ipv4_delete 从内核删除原来的路由；更改路由标志为 RIP_RTF_FIB;

- 如果新度量不为无穷大,使用 rip_timeout_update 函数重新计时。

■ **如果没有这样的路由，度量值也不是无穷大,则添加路由：**

- 设置 RTE 条目的目标前缀和长度;
- 设置度量为新计算的度量，标签为条目的标签;
- 设置下一跳地址(from)为从数据报来的路由器地址;
- 使用 rip_timeout_update 函数停止定时器；设置路由更改标志;
- 调用 rip_event (RIP_TRIGGERED_UPDATE,0);通知进程触发更新;
- 使用 rip_distance_apply 获得距离值;
- 使用 rip_zebra_ipv4_add 将路由下发到内核，更改路由标志。

第四部分综合实验

RIP 协议和转发引擎、路由管理、转发引擎性能的测试。

1 实验要求

第一个实验通过 RIP 协议下发路由和配置静态路由发送转发引擎，分别验证是否可以转发，第二个实验室是路由管理优先级的测试，第三个实验通过多条有效路由对转发引擎进行测试，验证引擎的转发速率。

2 实验内容

2.1 实验一：RIP 协议和转发引擎

（1）实验目的

通过实验验证静态路由的转发和 Rip 路由的转发。

（2）实验环境

如图 4.1 所示，学生机通过网络与测试服务器连接；利用学生机上的网卡 1 和网卡 2 与测试服务器相连接，测试服务器和学生机上运行路由协议 RIP。

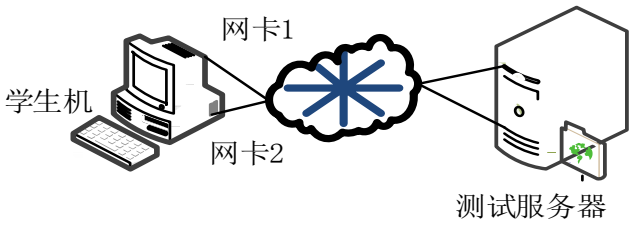

图 4.1 RIP 协议和转发引擎验证环境

（3）实验操作

步骤 1:搭建环境

- 在学生机上运行自己开发好的 RIP 协议,和测试服务器建立 RIP 连接。
- 在学生机上运行自己开发好的路由转发引擎。

步骤 2:将路由信息发送转发引擎

- 在 quagga 代码 rt_netlink.c 文件 netlink_route_multipath 函数中实现路由信息的发送。
- cmd 为 24 表示插入路由，为 25 表示删除路由。
- 创建 socket 套接字，连接转发引擎，发送路由信息。

步骤 3:验证静态路由

在学生机上配置一条静态路由，测试服务器向学生机发送 ping 报文，测试学生机的转发是否正常。如图 4.2 正常转发的现象:

```
localhost.localdomain# ping 192.168.5.2
PING 192.168.5.2 (192.168.5.2) 56(84) bytes of data.
发出的报文:
Source IP is 192.168.1.1
Destination IP is 192.168.5.2
TTL is 64
checksum is 55b3

收到的报文:
Source IP is 192.168.1.1
Destination IP is 192.168.5.2
TTL is 63
checksum is 55b4
checksum is right!!

From 192.168.1.4 icmp_seq=1 Destination Net Unreachable
```

图 4.2 转发正常的现象

如图 4.3 不能正常转发的现象:

```
发出的报文:
Source IP is 192.168.1.1
Destination IP is 192.168.5.2
TTL is 64
checksum is 55b3
From 192.168.1.4 icmp_seq=4 Destination Net Unreachable
发出的报文:
Source IP is 192.168.1.1
Destination IP is 192.168.5.2
TTL is 64
checksum is 55b3
From 192.168.1.4 icmp_seq=5 Destination Net Unreachable
没有收到包!!!!
From 192.168.1.4 icmp_seq=6 Destination Net Unreachable
没有收到包!!!!
From 192.168.1.4 icmp_seq=7 Destination Net Unreachable
没有收到包!!!!
From 192.168.1.4 icmp_seq=8 Destination Net Unreachable
没有收到包!!!!
```

图 4.3 不能正常转发现象

步骤 4:验证 RIP 路由

在学生机和测试服务器建立 RIP 连接，在测试服务器上配置一条静态路由，重分发此静态路由，学生机会通过 RIP 学到相应路由，在测试服务器向学生机发送 ping 报文，测试学生机的转发是否正常。

如图 4.4 学生机学到的重分发路由：

```
localhost.localdomain# sho ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

R>* 20.20.0.0/16 [120/2] via 192.168.1.4, eth6, 00:04:10
C>* 127.0.0.0/8 is directly connected, lo
C>* 192.168.1.0/24 is directly connected, eth6
R>* 192.168.3.0/24 [120/2] via 192.168.1.4, eth6, 00:25:15
R>* 192.168.5.0/24 [120/2] via 192.168.1.4, eth6, 00:25:12
C>* 192.168.7.0/24 is directly connected, eth8
```

图 4.4 RIP 重分发路由

转发是否正常借鉴上一个步骤

2.2 实验二：路由管理开发

（1）实验目的

通过实验熟悉路由优先级管理。

（2）实验原理

路由管理模块负责统一管理 RIP、OSPF、BGP 等模块学习到的路由信息，将这些模块学习到的路由信息存储在自己的路由表里，根据路由优先级的高低，选择相同路由信息中优先级最高的插入到转发表中；在向转发表插入表项时，查看是否有相同表项，当有路由表项相同时，原有的表项删除，选择优先级高的表项插入。

（3）实验环境

如图 4.5 所示，学生机通过网络与测试服务器连接；利用学生机上的网卡 1 和测试服务器相连接，测试服务器上运行路由协议 RIP。

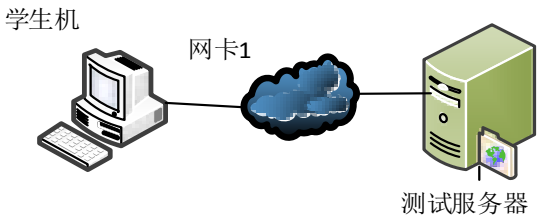

图 4.5 路由管理验证环境

（3）实验操作

步骤 1:开发管理模块

- 在 quagga 向转发引擎传输路由数据时将所需信息从原有结构体里填充到

发送数据里，实现函数位置:quagga 代码 rt_netlink.c 文件 netlink_route_multipath 函数中；

- 将接收到的路由信息插入转发表，当有相同转发表项时，进行优先级的比较，若原有的表项优先级高，不插入当前表项；若当前表项优先级高，则删除原来的表项，重新插入当前表项；
- 在匹配选中转发表中的表项时，添加打印信息，将路由优先级打印出来。

步骤 2:验证路由优先级管理

在学生机和测试服务器上建立 RIP 连接，然后在学生机上配置一条静态路由，这条路由和学生机通过 RIP 学到的路由一样，测试服务器向学生机发送 ping 报文，要求将打印的路由优先级变量信息展示。

2.3 实验三：转发性能测试

（1）实验环境

如图 4.6 所示，学生机通过网络与测试仪连接；利用学生机上的网卡 1 发送数据包，网口 2 接收数据包，来测试转发的最大速率。

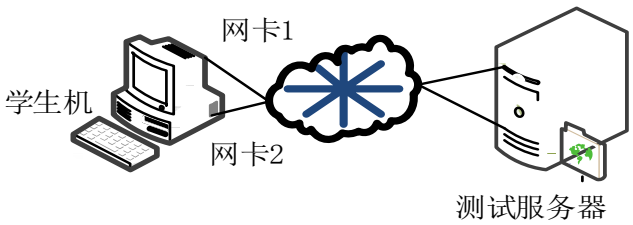

图 4.6 转发性能验证环境

（2）实验操作

步骤 1:搭建测试环境

- 学生机和测试服务器上都运行 RIP，与测试服务器相连接。
- 学生机上运行转发引擎。
- 学生机上运行客户端程序，连接转发引擎。

步骤 2:向转发引擎发送多条路由信息，进行验证

- 在测试服务器上随机配置一条静态路由，如图所示，触发读取路由表项，测试服务器会循环读取存储路由表项的文件，下发多条路由表项，学生机通过 RIP 学到这些路由并下发到转发引擎。

如图 4.7 测试服务器读取路由表项并下发:

```
R7(config)# ip route  192.168.5.0/24 192.168.3.2 1
% Malformed address
R7(config)# end
R7# sho ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

C>* 127.0.0.0/8 is directly connected, lo
C>* 192.168.1.0/24 is directly connected, GE1
C>* 192.168.3.0/24 is directly connected, GE2
R>* 192.168.6.0/24 [120/2] via 192.168.1.3, GE1, 00:03:43
S>* 192.168.104.0/24 [1/0] via 192.168.3.2, GE2
S>* 192.168.105.0/24 [1/0] via 192.168.3.2, GE2
S>* 192.168.106.0/24 [1/0] via 192.168.3.2, GE2
S>* 192.168.107.0/24 [1/0] via 192.168.3.2, GE2
S>* 192.168.108.0/24 [1/0] via 192.168.3.2, GE2
S>* 192.168.109.0/24 [1/0] via 192.168.3.2, GE2
S>* 192.168.110.0/24 [1/0] via 192.168.3.2, GE2
S>* 192.168.111.0/24 [1/0] via 192.168.3.2, GE2
S>* 192.168.112.0/24 [1/0] via 192.168.3.2, GE2
S>* 192.168.113.0/24 [1/0] via 192.168.3.2, GE2
S>* 192.168.114.0/24 [1/0] via 192.168.3.2, GE2
```

图 4.7 下发路由表项

如图 4.8 学生机通过 RIP 学到测试服务器发送的多条路由表项：

```
R4# sho ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

C>* 127.0.0.0/8 is directly connected, lo
C>* 192.168.1.0/24 is directly connected, GE1
R>* 192.168.3.0/24 [120/2] via 192.168.1.2, GE1, 00:04:22
C>* 192.168.6.0/24 is directly connected, GE2
R>* 192.168.104.0/24 [120/2] via 192.168.1.2, GE1, 00:00:16
R>* 192.168.105.0/24 [120/2] via 192.168.1.2, GE1, 00:00:15
R>* 192.168.106.0/24 [120/2] via 192.168.1.2, GE1, 00:00:15
R>* 192.168.107.0/24 [120/2] via 192.168.1.2, GE1, 00:00:15
R>* 192.168.108.0/24 [120/2] via 192.168.1.2, GE1, 00:00:15
R>* 192.168.109.0/24 [120/2] via 192.168.1.2, GE1, 00:00:15
R>* 192.168.110.0/24 [120/2] via 192.168.1.2, GE1, 00:00:15
R>* 192.168.111.0/24 [120/2] via 192.168.1.2, GE1, 00:00:15
R>* 192.168.112.0/24 [120/2] via 192.168.1.2, GE1, 00:00:15
R>* 192.168.113.0/24 [120/2] via 192.168.1.2, GE1, 00:00:15
R>* 192.168.114.0/24 [120/2] via 192.168.1.2, GE1, 00:00:15
R>* 192.168.115.0/24 [120/2] via 192.168.1.2, GE1, 00:00:15
R>* 192.168.116.0/24 [120/2] via 192.168.1.2, GE1, 00:00:15
R>* 192.168.117.0/24 [120/2] via 192.168.1.2, GE1, 00:00:15
```

图 4.8 学到的路由表项

- 文件中存储的路由表项下一跳都是 192.168.3.2，所以要求学生机的网卡 2 的 ip 为 192.168.3.0/24 网段。

- 文件 route10 中有 10 条路由，随机一条路由通过测试软件来验证转发速率。
- 文件 route100 中有 100 条路由,随机一条路由通过测试软件来验证转发速率。

步骤 3:验证转发速率

- 通过测试软件设置一个转发速率，查看发送报文和接收报文的个数大小。
- 如果发送报文的个数大小等于接收报文的个数大小，说明可以达到这个转发速率，继续设置更大的转发速率，以此来验证最大的转发速率。

运行测试转发性能程序(sendether.c)的结果:

```
[root@localhost home]# ./a.out
please input the ipv4 address :
192.168.105.2
please input the speed(one second how much packet send) :
1000
^Csend packets number is 2401
send packet speed is  0.197Mbps
recv packets number is 2084
send packet speed is  0.171Mbps
[root@localhost home]#
```