路由器实验报告

个人信息和实验环境

杨雅儒，2017011071 。

在 Ubuntu18.04 上进行代码编写和仿真测试，在 Raspbian(Debian 10) 上进行实验。

思路和算法

概览

RIP协议的实现参考了课程 github 上的 README.md，以及 RFC2453。主要实现的部分有：

- rip 的**请求**功能和**响应**功能，以及**路由转发**功能；
- **触发更新**机制——为了方便，并没有采用RFC1624中的延迟更新计时器，而是直接发出更新包，将路由表中得到更新的部分以响应的方式发送给邻居；
- **定时响应**机制——每过30秒，以响应的方式主动向邻居发送自己的路由表；
- **毒性逆转**机制——在发送响应时，若某个路由表项来自该接口，则往该接口发送时将此表项的metric赋值为16，表示无穷远，有利于提高收敛速度；
- **校验和增量计算**。

请求和响应

在路由器启动时，发出一次请求和响应。发出请求是为了获取邻居的路由表信息，发出响应是为了将自己的直连路由信息告知邻居。之后：

- 每过30秒主动向所有邻居发送一次响应；
- 在接收到请求时向发出请求的邻居发送响应；
- 接收到响应时若有更新，则向所有邻居发送更新部分的路由表项对应的响应。

请求报文的封装与发送

从高到低依次为 RIP -> UDP -> IP，故流程如下：

- 封装 RIP 请求，command 字段为 1（表示请求），version 字段为 2，Metric 字段为 16，其余为 0；
- 封装 UDP 头：源、目的端口均为 520；
- 封装 IP 头：源 ip 为本机网口 ip，目的 ip 为组播 ip 地址 224.0.0.9——该地址表示所有RIPv2路由器；
- 对于每个网口都进行封装之后通过 HAL 发送出去。

接收到请求报文的处理

- 判断 Metric 是否为 16，地址族标识是否为 0；
- 获取源 ip 地址；
- 遍历 RIP 路由表，封装所有与接收接口不为同一网段的路由表项到 RIP 报文里，并使用**毒性逆转**，对于出接口与当前接口相同的路由表项，发送的 matric 为 16。注意 RIP 报文的 command 字段为2。
- 封装 UDP 头和 IP 头，通过 HAL 从接收到请求的网口发送出去。

接收到响应报文的处理——后接更新报文的封装与发送

- 首先若没有到该网段的路由，且 metric < 15，则直接插入；
- 若原本已经有到该网段的路由表项了，则判断表项中的 nexthop 是否同来源ip相同，若相同则直接进行更新——若metric+1>=16则进行删除，若metric+1<16则用metric+1更新表项；若不相同则仅当 metric + 1 小于表项的 metric 时才更新。
- 接下来进行**触发更新**，对于所有更新之后的路由表项（注意需要包括被删除的路由），进行封装之后发送出去，注意使用**毒性逆转**。

转发

下面假定从网卡 1 转发到网卡 2：

- 通过 HAL 获取 IP 数据包；
- 对 IP 数据包进行验证（校验和等）；
- 取出 IP 数据包中的目的 IP 地址，并查找转发表，获取出口接口以及 nexthop；
- 对 IP 数据包进行修改，并重新计算校验和，再封装成新的 IP 数据包；
- 若 nexthop 为 0.0.0.0，表明出口接口与目的ip地址处于同一网段，则此时通过 HAL 查询 ARP 获得目的 ip 地址对应的 MAC 地址；若 nexthop 不为 0.0.0.0，则通过 HAL 查询 ARP 表获取 nexthop 的 MAC 地址；
- 通过 HAL 向该 MAC 地址发送封装的 ip 数据包。

校验和的增量计算

校验和增量计算参考了 RFC1624，令：

- HC：旧的校验和；
- C：旧的1补码和，即反码和；（注意是按照每16位作为一个反码）
- HC'：新的校验和；
- C'：新的1补码和；
- m：某个16位的旧值；
- m'：该16位的新值。

首先，HC = ~C，这是由校验和的定义决定的。在一开始计算校验和的过程中，**溢出则移位相加实际上便是在求反码和C**，最后取反则得到 HC。

那么有：

$$HC' = \sim (C + (-m) + m') = HC + (m - m') = HC + m + (\sim m')$$

最后还有一个小问题——**反码中的正零和负零**，注意到我们最先计算校验和的过程中，实际上保证了补码和只会出现负零，也就是保证了校验和只会出现全 0 而不会出现全 1。所以在上式的基础上，对 HC' 进行判断——若全1，则更改为全0 。

仿真测试

在 Ubuntu 18.08 上，通过 netns 进行。在 test\ 下，有test\test1，test\test2 和 test\test3 为三个类型的测试文件夹，其中 test\test3\test3_small、test\test3\test3_medium、test\test3\test3_large 分别对应小规模、中规模、大规模压测。

测试方法

每个文件夹对应一组测试。

首先运行 createNetns.sh 创建网络命名空间 netns。

接下来两个选择——运行 runBird.sh，将打开在三个 netns 中各运行一个 bird【这个方法的测试代码后期并未继续维护，可认为弃置，不过在runMyTest.sh的基础上稍作修改即可使用】；运行 runMyTest.sh，将打开两个 bird 和一个自己实现的路由器程序。

最后可以运行 deleteNetns.sh 删除这些网络命名空间。

目录

test1

同拓扑图中仅含 R1、R2、R3 的部分。

```
netns名称:: 接口1名称: 接口1IP  [接口2名称: 接口2IP ...]
net1:: veth-1r: 192.168.3.1
net2:: veth-2l: 192.168.3.2  veth-2r: 192.168.4.1
net3:: veth-3l: 192.168.4.2
```

test2

同拓扑图，在上面的基础上加入了两端 net0 和 net4，作为两端的机器。

```
netns名称:: 接口1名称: 接口1IP  [接口2名称: 接口2IP ...]
net0::                        veth-0r: 192.168.2.1
net1:: veth-1l: 192.168.2.2   veth-1r: 192.168.3.1
net2:: veth-2l: 192.168.3.2   veth-2r: 192.168.4.1
net3:: veth-3l: 192.168.4.2   veth-3r: 192.168.5.1
net4:: veth-4l: 192.168.5.2
```

另外在 test2 中，将路由器更新时间全部设置为 30s，而不是原来的 5s。

test3

在 test2 的基础上，加入了最终要求的压测。

test3_small、test3_medium、test3_large分别对应小规模、中规模、大规模压测。

实验

按照要求进行了实验，拓扑上按照附加分要求连成了环，并且在第三部分获得了 195.2 的高分。

调试和经验

关于路由转发

- 注意到 **Ubuntu 自带了路由转发功能**，在运行 bird 的 netns 上需要开启该路由转发：

```
ip netns exec net1 sysctl -w net.ipv4.ip_forward=1
```

- 但是在**运行自己实现的路由器的 netns 上不能开启路由转发**！否则会重复转发，导致出现重复包。

关于树莓派与Ubuntu的网络共享

这部分也花了不少时间才成功进行网络共享。

树莓派静态IP设置

使用 vim 或者 nano 打开 /etc/dhcpcd.conf 文件，按照其中注释的格式在末尾条目即可，例如：

```
interface eth0
static ip_address=192.168.11.2/24
static routers=192.168.11.1
static domain_name_servers=192.168.11.1 8.8.8.8
```

然后使用如下指令即可生效：

```
service dhcpcd restart
```

Ubuntu18.04 静态IP 设置

在 /etc/netplan/*.yaml 设置例如（其中 enp0s25 为一个网络接口）：

```
network:
  version: 2
  renderer: NetworkManager
  ethernets:
    enp0s25:
      addresses: [192.168.11.1/24]
      nameservers:
        addresses: [114.114.114.114, 8.8.8.8]
```

最后使用指令 `netplan apply` 应用修改。

Ubuntu 路由转发和 NAT 设置

如下设置 Ubuntu 的路由转发和 NAT：

```
sysctl -w net.ipv4.ip_forward=1
iptables -t nat -A POSTROUTING -o wlp4s0 -j MASQUERADE
```

至此，树莓派终于可以连接互联网。

关于两侧 PC1 和 PC2 的网关设置

一开始 PC1 和 PC2 ping 不出去，通过抓包才发现是没有配置网关导致的，通过 ip route 配置一下默认网关为直连的路由器即可。

关于 Checksum offload

Checksum offload 的目的是减少 CPU 资源的消耗，将这部分工作交由硬件完成，若开启了 Checksum offload，将导致抓到的包校验和显示不正确，例如使用如下指令即可关闭veth-1r网口的tx offload：

```
ethtool -K veth-1r tx off
```

关于回环设备

注意在新建的 netns 中，回环接口 lo 默认是关闭状态，使用如下指令可以打开：

```
ip netns exec net0 ip link set lo up
```

鸣谢

感谢队友的共同努力。

感谢助教们的辛苦付出。

感谢老师的认真教导。

完结撒花。