RIP 协议实验报告

计 13 谭志鹏 2011011258

一、 实验目的

RIP（Routing Information Protocol）协议是目前互联网上最简单的一种路由协议。RIP协议采用距离向量路由算法。本实验通过在NetRiver实验系统上实现RIP协议，加深对路由协议基本原理和距离向量算法的理解，了解RIP协议的分组接收和发送流程以及路由表的维护，进一步掌握计算机网络中的核心技术——路由技术，并培养路由协议编程开发能力。

二、 实验原理

1、 RIP 协议

交互的RIP协议信息分组主要是两种类型：请求（request）分组和响应（response）分组。请求分组用来向相邻运行RIP协议的路由器请求路由信息，响应分组用来发送本地路由器的路由信息。RIP协议使用距离向量路由算法，因此发送的路由信息可以用<vector, distance>来表示。在实际分组中，vector用路由的目的地址address表示，而distance用该路由的距离度量值metric表示。metric值规定为从本机到达目的网络路径上经过的路由器数目，metric的有效值为1～16，其中16表示网络不可到达

当发出请求的路由器接收到一个 response 分组后，它会逐一处理收到的路由表项内容。如果分组中的表项为新的路由表项，那么就会向自身路由表中加入该表项。如果该分组中的表项已经在路由表中存在，那么首先判断这个收到的路由更新信息是从哪个路由器发送过来的：如果就是从这个表项的源路由器（即当初发送相应路由信息从而导致产生这个路由表项的路由器）发来的，则无论该现有表项的距离度量值（metric）如何，都需要更新该表项；如果不是，那么只有当更新表项的 metric 值小于路由表中相应表项 metric 值时才需要替代原来的表项。

为了保证路由的有效性，RIP 协议规定：每隔 30s，重新广播一次路由信息；若连续三次没有收到邻居路由器广播的 RIP 路由信息，则与这些路由器相关的路由信息失效。

水平分裂算法是一种避免路由回路和加快路由收敛的技术。由于路由器可能收到它自己发送的路由信息，而这种信息是无用的。水平分裂算法规定，路由器不反向通告任何从邻居收到的路由更新信息。

2、 RIP 分组信息

RIPv2 的分组结构如图 3-15 所示。每个 RIPv2 分组都由 RIP 头部（RIP Header）和最多 25 个 RIP 路由项（RIP Entry）组成。如果路由表的路由表项数目大于 25 时，就需要多个 RIP 分组来完成路由信息的传播过程。由图可知，RIP 头部包括 Command、Version、must be zero 三个字段，一个 RIP 路由项包括 Address Family Identifier、Route Tag、IP Address、Subnet Mask、Next Hop 和 Metric 等六个字段。可以推算出 RIPv2 分组的最大长度为 512B，其中包括 8B 的 UDP 头部、4B RIP 头部和最多 500B（20B25）路由项。具体如下：

Command：表示RIP分组的类型，目前RIP只支持两种分组类型，分别是请求分组

（request 1）和响应（response 2）分组。

Version：表示RIP分组的版本信息，RIPv2分组中此字段为2。

must be zero：保留字段，取值为0。

Address Family Identifier：表示路由信息所属的地址族，目前RIP中规定此字段必须为2，表示使用IP地址族。

IP Address：表示路由信息对应的目的IP地址，可以是网络地址、子网地址或主机地址。

Subnet Mask：子网掩码，应用于上述IP地址产生非主机地址，为0时表示不包括子网掩码部分，使得RIP能够适应更多的环境。

Next Hop：下一跳IP地址，可以对使用多路由协议的网络环境进行路由优化。

Metric：表示从本路由器到达目的地的距离，目前 RIP 将路由路径上经过路由器的个数作为距离度量值。

分组结构具体如图：

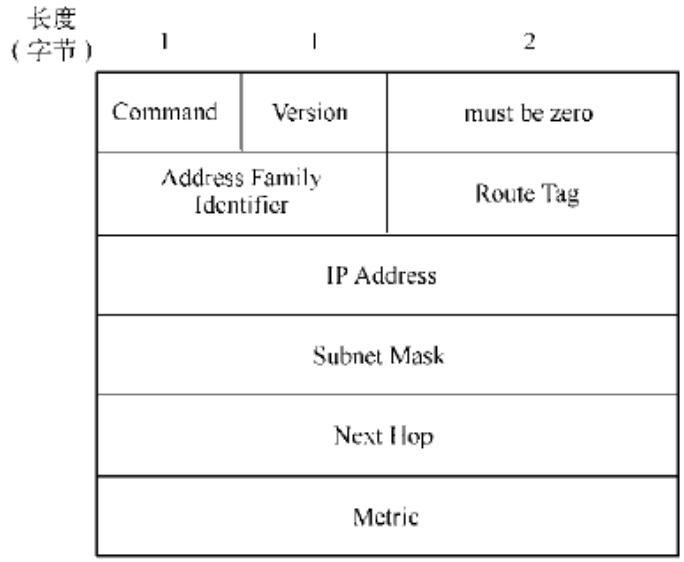

图 3-15　RIPv2 的分组结构

三、　实验内容

1、 实验分析：

实验中需要完成的内容具体包括：

1）对客户端接收到的 RIP 报文进行有效性检查：对客户端接收到的 RIP 协议报文进行合法性检查，丢弃存在错误的报文并指出错误原因。

2）处理 Request 报文：正确解析并处理 RIP 协议的 Request 报文，并能够根据报文的内容以及本地路由表组成相应的 Response 报文，回复给 Request 报文的发送者，并实现水平分割。

3）处理 Response 报文：正确解析并处理 RIP 协议的 Response 报文，并根据报文中携带的路由信息更新本地路由表。

4）路由表项超时删除：处理来自系统的路由表项超时消息，并能够删除指定的路由。

5）路由表项定时发送：实现对本地路由的定时发送功能，并实现水平分割。

实验中需要实现的接口函数：

1）RIP 分组处理函数： int stud_rip_packet_recv(char *pBuffer, int bufferSize, UINT8 iNo, UINT32 srcAdd)

2）RIP 超时处理函数： void stud_rip_route_timeout(UINT32 destAdd, UINT32 mask, unsigned char msgType)

2、 程序分析：

（1） RIP 分组处理函数

首先检查 RIP 分组是否合法，若分组有错误则报错

```
int stud_rip_packet_recv(char *pBuffer, int bufferSize, UINT8 iNo, UINT32 srcAdd)
{
    RipPacket* ripPacket = (RipPacket*)pBuffer;
    //检查数据包的头部是否合法，检查是否有版本错误或者命令错误
    if(ripPacket->rip_vers != 2)
    {
        ip_DiscardPkt(pBuffer, STUD_RIP_TEST_VERSION_ERROR);
        return -1;
    }
    if(ripPacket->rip_cmd != 1 && ripPacket->rip_cmd != 2)
    {
        ip_DiscardPkt(pBuffer, STUD_RIP_TEST_COMMAND_ERROR);
        return -1;
    }
```

根据 command 域，分 request 和 response 两个分组进行处理。

处理 request:

根据本地路由表信息组成 Response 分组，并通过 rip_sendIpPkt()函数发送出去

```
//分为command域，  分request 和response来响应
if(ripPacket->rip_cmd == 1)   // request，水平分裂发送本地路由表信息
{
    sendPacket(iNo);
}
```

其中 sendPacket 为一个子函数，用于水平分裂发送本地路由表信息之 iNo 端口。由于这部分程序在后面的处理中也有涉及，写成一个子函数比较方便。

```
void sendPacket(int iNo)  //水平分裂发送本地路由表信息到iNo端口
{
    stud_rip_route_node *pnode = g_rip_route_table;
    RipPacket res;
    res.rip_cmd = 2;      //头部封装，发送response
    res.rip_vers = 2;
    res.rip_mbz = 0;
    int send_count = 0;
    while(pnode != NULL)   //遍历路由表
    {
        if (pnode->if_no != iNo)  //水平分裂，不发送该来源接口的路由信息
        {
            res.rip_rts[send_count].rr_family = htons(2);
            res.rip_rts[send_count].rr_rttag = 0;
            res.rip_rts[send_count].rr_addr.u_l = htonl(pnode->dest);
            res.rip_rts[send_count].rr_mask.u_l = htonl(pnode->mask);
            res.rip_rts[send_count].rr_nexthop.u_l = htonl(pnode->nexthop);
            res.rip_rts[send_count].rr_metric = htonl(pnode->metric);
            ++send_count;
        }
        pnode = pnode->next;
    }
    if(send_count > 0){       //调用rip_sendIpPkt发送路由信息，  发送长度为头部加体的长度
        rip_sendIpPkt((unsigned char*)(&res), 4 + 20 * send_count, 520, iNo);
    }

}
```

处理 response：

首先提取出该分组中携带的所有路由表项，并遍历本地路由表查找是否存在该表项。

```
for(int i = 0; i < 16; i++)
{
    stud_rip_route_node *pnode = g_rip_route_table;
    stud_rip_route_node *prev = NULL;

    RipRt q = ripPacket->rip_rts[i];

    if (q.rr_addr.u_l == 0) break;
    int dest = ntohl(q.rr_addr.u_l);
    int mask = ntohl(q.rr_mask.u_l);
    int metric = ntohl(q.rr_metric);
    int nexthop = ntohl(q.rr_nexthop.u_l);

    while(pnode != NULL
       && !(pnode->dest == dest
       && pnode->mask == mask))
    {
        prev = pnode;
        pnode = pnode->next;
    }
```

根据每个路由表项分三种情况处理：

情况一：若该 Response 路由表项为新的表项，即该 Response 路由表项中的 IP Address 与所有本地路由表项的 IP Address 都不相同，则将其 Metric 值加 1 之后添加到本地路由表中。如果 Metric 加 1 之后等于 16，则表明该 Response 路由表项已经失效，此时无需添加该表项。

```
//1：遇到新的路由表项，距离加1后若小于16
if(pnode == NULL)
{
    if(metric == 15) continue;
    pnode = new stud_rip_route_node;
    pnode->dest = dest;
    pnode->mask = mask;
    pnode->nexthop = srcAdd;
    pnode->metric = metric + 1;
    pnode->if_no = iNo;
    pnode->next = NULL;
    if(g_rip_route_table == NULL)
        g_rip_route_table = pnode;
    else
        prev->next = pnode;
}
```

情况二：若该 Response 路由表项已经在本地路由表中存在，且二者的 Next Hop 字段相同，则将其 Metric 值加 1 后更新到本地路由表项的 Metric 字段。如果 Metric 加 1 之后等于 16，应置该本地路由表项为无效。

```
//2：本地路由表中存在，且next字段相同
else if(pnode -> nexthop == srcAdd)
{
    pnode->metric = metric + 1;
    pnode->if_no = iNo;
}
```

情况三：若该 Response 路由表项已经在本地路由表中存在，且二者的 Next Hop 字段不同，则只有当 Response 路由表项中的 Metric 值小于本地路由表项中的 Metric 值时，才将其 Metric 值加 1 后更新到本地路由表项的 Metric 字段，并更新 Next Hop 字段。如果 Metric 加 1 之后等于 16，应置该本地路由表项为无效。

```
//3：本地路由表中已经存在，next字段不同，需要比较距离值，更优是更新
else if(metric < pnode->metric)
{
    pnode->metric = metric + 1;
    pnode->if_no = iNo;
    pnode->nexthop = srcAdd;
}
```

（2） RIP 超时处理函数

根据 msgType 分两种情况处理：

若 msgType 为 RIP_MSG_SEND_ROUTE 则进行路由信息广播，应该在每个接口上分别广播自己的 RIP 路由信息。其中 sendPacket 函数上面已经说明。

```
//路由广播信息，分别在端口1和2发送一遍路由信息
if (msgType == RIP_MSG_SEND_ROUTE)
{
    for(int i = 1; i <= 2; i++)
    {
        sendPacket(i);
    }
}
```

若 msgType 为 RIP_MSG_DELE_ROUTE，说明计时器超时。应遍历本地路由表找到超时的表项，置该项为无效，即 metric 值置为 16。

```
//某个路由计时器超时，该路由表项设为无效
else {
    stud_rip_route_node *pnode = g_rip_route_table;
    while(pnode != NULL
          && !(pnode->dest == destAdd
          && pnode->mask == mask))
    {
        pnode = pnode->next;
    }
    if(pnode != NULL)
    {
        pnode->metric = 16;
    }
}
```

四、 思考题

1、 因为协议中定义了一旦跳数达到16，此路径被声明为不可用的，从而将此路径从路由表中剔除。如果不这么做的话一旦形成环路的话将无限制地路由下去，造成网络瘫痪。

2、 RIP是距离矢量路由选择协议，只是用跳数来决定到达远程网络的最佳方式，并且每隔30秒就送出自己完整的路由表到所有的接口，所以RIP协议在小型的网络中会运转良好，在使用慢速WAN链接的大型网络或安装了大量路由器的网络来说，效果略显不足。OSPF(开放最短路径优行)是开放标准的路由选择协议，工作原理主要为Dijkstra算法，先构建最短路径树，然后使用最佳路径的结果来组件路由表。因此OSPF会更快，支持到达相同目标的多个等开销路由，所以在大型的网络上，

OSPF占有优势。

五、 总结

本次试验一开始进行了 RIP 协议交互试验。通过实际观察报文，手工进行处理，对于 RIP 的报头更加熟悉。这些题目也对于里接 RIP 协议有很好的帮助，这样使得后续的编程试验更加的顺利。

本次试验中一开始对于水平分裂算法的理解存在一定的偏差，对于端口这个概念没有理解充分。水平分裂算法时，在发送本地路由表项时不发送来源接口的路由信息，而实验中只有两个接口，若接口 1 有请求，则应将本地路由中接口 2 的路由表项发送到接口 1。

另外一个问题就是处理 response 时，在更新表项时应该将 srcAdd 赋给本地表项，而不是将数据包中的目标地址复制过去。

这次实验过程中遇到问题时及时的和同学和助教进行讨论，这样解决问题比较快。通过这次实验，对于路由算法有了更多的理解。