09-网络路由实验

范子墨
2021K8009929006

一、 实验内容

a) 实验一
   i. 基于已有代码框架，实现路由器生成和处理 mOSPF Hello/LSU 消息的相关操作，构建一致性链路状态数据库
   ii. 运行实验
      1. 运行网络拓扑(topo.py)
      2. 在各个路由器节点上执行 disable_arp.sh, disable_icmp.sh, disable_ip_forward.sh)，禁止协议栈的相应功能
      3. 运行./mospfd，使得各个节点生成一致的链路状态数据库

b) 实验二
   i. 基于实验一，实现路由器计算路由表项的相关操作
   ii. 运行实验
      1. 运行网络拓扑(topo.py)
      2. 在各个路由器节点上执行 disable_arp.sh, disable_icmp.sh, disable_ip_forward.sh)，禁止协议栈的相应功能
      3. 运行./mospfd，使得各个节点生成一致的链路状态数据库
      4. 等待一段时间后，每个节点生成完整的路由表项
      5. 在节点 h1 上 ping/traceroute 节点 h2
      6. 关掉某节点或链路，等一段时间后，再次用 h1 去 traceroute 节点 h2

二、 总体设计思路

基于上一个实验，对数据包进行进一步的分类处理。若目的地址与接口地址匹配，则增加判断上层协议是否是 IPPROTO_MOSPF 字段，将其交给 handle_mospf_packet 函数进行处理。若目的地址与接口地址不匹配，则判断是否为等于多播 OSPF 所使用的 IP 地址 MOSPF_ALLSPFRouters，若等于，则断言其为多播 OSPF 数据包，同样交由 handle_mospf_packet 函数进行处理。如果目的 IP 地址既不等于接口 IP 地址，也不等于 MOSPF_ALLSPFRouters，则说明这个数据包需要进行路由转发，将其交给 ip_forward_packet 函数进行进一步处理。这里需要自己将上个实验中对路由转发的处理代码复制到 ip_forward 函数中。

在 handle_mospf_packet 函数中，首先检验该数据包是否有错误，而后根据 mospf->type 进行分类。如果是 MOSPF_TYPE_HELLO 类型，则为 hello 数据包，跳转至 handle_mospf_hello。若是 MOSPF_TYPE_LSU 类型数据包，则调用 handle_mospf_lsu 函数处理。

handle_mospf_lsu 会处理三类 lsu 消息，包括新的链路状态信息、原有状态信息的定期发送、已记录或过期的状态信息。根据数据包，对本路由器的链路状态数据库进行更新。而后由于洪泛链路，因此在处理好消息后，如果 TTL 值还大于 0，向除本端口以外的端口转发消息，一层层传递下去。

与此同时，每个节点需要周期性发送 MOSPF Hello（ hello-interval ：5 秒）宣告自己的存在，同时周期性发送 MOSPF LSU 数据包。此外，要定期检查邻居列表和 MOSPF 数据库表。由于本次实验没有默认路由表，因此需要根据 Dijkstra 算法进行路由表的生成。即同时创造以下线程，同时运行。

```
pthread_create(&hello, NULL, sending_mospf_hello_thread, NULL);
pthread_create(&lsu, NULL, sending_mospf_lsu_thread, NULL);
pthread_create(&nbr, NULL, checking_nbr_thread, NULL);
pthread_create(&db, NULL, checking_database_thread, NULL);
pthread_create(&rtable, NULL, generate_rtable_thread, NULL);
```

三、 生成和处理 mOSPF Hello/LSU

version	type	length
router id		
area id		
checksum		padding

图表 1：mOSPF 表头

1、 发送 mOSPF Hello

遍历路由器的接口列表，最后将构造好的数据包通过接口发送给邻居，数据包赋值过程与上一个实验类似。需要注意由于 mOSPF 为多播，因此发包的目的 mac 地址为 01:00:5E:00:00:05，ip 地址为 224.0.0.5。每次循环前，需要先休眠一段时间，

```
u8 dhost[ETH_ALEN] = {0x01, 0x00, 0x5e, 0x00, 0x00, 0x05};
eh->ether_type = htons(ETH_P_IP);
memcpy(eh->ether_shost, iface->mac, ETH_ALEN);
memcpy(eh->ether_dhost, dhost, ETH_ALEN);
```

mask	
hello interval	padding

图表 2：Hello 数据包

2、 处理 mOSPF Hello

收到 mOSPF Hello 消息后，对数据包进行处理。首先遍历邻居列表，检查是否有邻居 ID 与收到的 ID 相匹配，若匹配，则设置 found=1，表示该数据包处理完成，同时设置该条目的 alive 计数器为 0。若未找到匹配，则表明邻居列表中需要新增条目，按照 PPT 上的数据结构进行构建即可，最后将该邻居添加到该接口的邻居列表中，由于信息发生变动，因此向其他邻居发送 lsu 信息。

```
typedef struct {
    struct list_head list;
    u32     nbr_id;          // neighbor ID
    u32     nbr_ip;          // neighbor IP
    u32     nbr_mask;        // neighbor mask
    u8      alive;           // alive for #(seconds)
} mospf_nbr_t;
```

3、 定期检查邻居列表

首先遍历该路由器的所有接口，然后遍历该接口的所有邻居，将 alive 实践增加，如果该邻居已经超过 15s 没有更新过，则删除。

```
if (nbr->alive > 3 * iface->helloint)
{
```

```
        list_delete_entry(&nbr->list);
        free(nbr);
        iface->num_nbr--;
        sending_mospf_lsu();
    }
```

4、 生成 mOSPF LSU

首先是在 sending_mospf_lsu 函数中定期发送 lsu 信息，即每 40s 跳转到 sending_mospf_lse 函数中。

在 sending_mospf_lsu 函数中，首先遍历路由器的接口，得到该路由器的总邻居数。而后创建数组 lsa，填充每个接口的邻居信息，包括掩码、网络地址和路由器 ID。

```
    list_for_each_entry (iface, &instance->iface_list, list)
    {
        if (iface->num_nbr == 0)
        {
            lsa_array[i].mask = htonl(iface->mask);
            lsa_array[i].network = htonl(iface->ip & iface->mask);
            lsa_array[i].rid = 0;
            i++;
        }
        else
        {
            mospf_nbr_t *ptr = NULL;
            list_for_each_entry (ptr, &iface->nbr_list, list)
            {
                lsa_array[i].mask = htonl(ptr->nbr_mask);
                lsa_array[i].network = htonl(ptr->nbr_ip & ptr->nbr_mask);
                lsa_array[i].rid = htonl(ptr->nbr_id);
                i++;
            }
        }
    }
```

而后更新该路由协议的序列号。

最后再次遍历该接路由器的所有邻居，将 lsu 数据包赋值并发给所有邻居。处理方式与 hello 数据包类似。

sequence number	ttl	unused
#(advertisement)		
network		
mask		
router id		
… …		

图表 3：lsu 数据包

5、 处理 mOSPF LSU

将受到的数据包拆分出 ip 首部、mospf 首部、lsu 部分和 lsa 部分。而后遍历数据链路库，如果存在匹配的路由器 rid，则设置 found=1，如果序列号大于原序列号，则更新当前路由器的所有邻居信息。如果没有找到匹配的条目，则创建一个新的数据库条目，填充 RID、序列号、邻居数目，并为每个 LSA 信息分配内存并填充。

```
mospf_db_entry_t *new_db = (mospf_db_entry_t *)malloc(sizeof(mospf_db_entry_t));
new_db->rid = ntohl(mospf_header->rid);
new_db->seq = ntohs(lsu->seq);
new_db->nadv = ntohl(lsu->nadv);
new_db->alive = 0;
new_db->array = (struct mospf_lsa*)malloc(new_db->nadv * MOSPF_LSA_SIZE);

for (int i = 0; i < new_db->nadv; i ++ )
{
    new_db->array[i].network = ntohl(lsa[i].network);
    new_db->array[i].mask = ntohl(lsa[i].mask);
    new_db->array[i].rid = ntohl(lsa[i].rid);
}
list_add_tail(&new_db->list, &mospf_db);
```

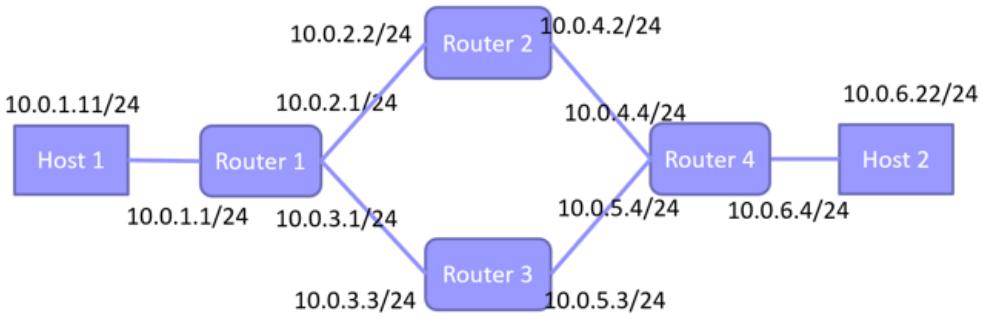
将 ttl-1，如果 ttl 还大于 0，将该数据包更改 ip 首部和以太网首部后再次发出。

```
iph->saddr = htonl(iface->ip);
iph->daddr = htonl(nbr->nbr_ip);

mospfh->checksum = mospf_checksum(mospfh);
iph->checksum = ip_checksum(iph);

ip_send_packet(forward, len);
```

6、 定期检查 mOSPF 数据库表

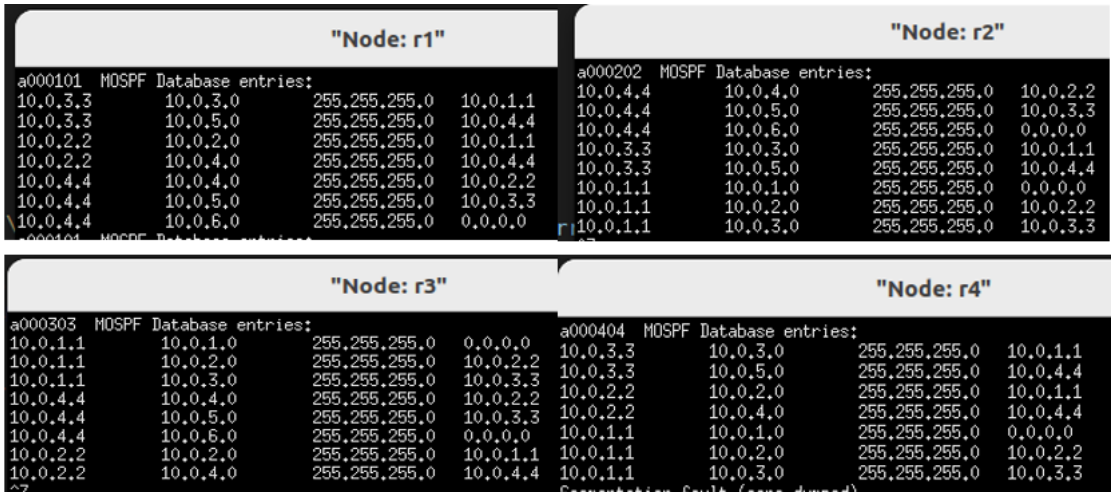
遍历数据链路库中的所有条目，如果数据库中一个节点的链路状态超过 40 秒未更新时，表明该节点已失效，将对应条目删除。

7、 结果



图表 4：拓扑

可以看出根据上图，数据链路库生成正确。



图表 5：链路状态数据库

四、 实现路由器计算路由表项的相关操作

1、 生成路由表

首先初始化一个图数据结构，在 init_graph 函数中根据链路状态数据库中的信息建立路由器中的拓扑图，而后执行 Dijkstra 算法，找到最短路径的下一个节点，将其标记为已访问。

而后遍历路由表，如果路由表项有下一跳地址，则移除该路由表项，即每个路由器的路由表只有其端口所连接的局域网。

对于每个节点，会根据 Dijkstra 算法匹配到前序节点。用递归的方式前递可以找到对于本节点而言，每一个其他的节点的下一条节点是多少，并以此更新路由表，从而确定到其他网络的下一跳网关地址、源节点的转发端口。

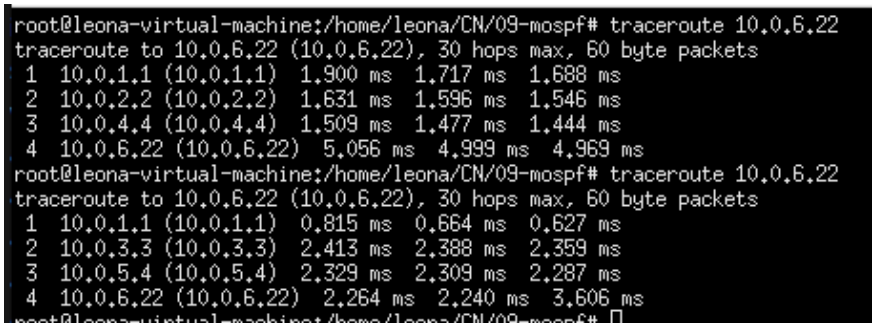
2、 Dijkstra 算法

伪代码在讲义中已经给出，实现思路为首先初始化 dist 数组（dist[i] 表示从起点到节点 i 的最短距离），prev 数组（prev[i] 表示节点 i 的前驱节点），visited 数组（标记节点是否已经被访问）。使用外循环遍历每一个路由器，在每次循环中，找到距离起点距离最短且未被访问的节点，这里使用 min_dist 函数实现。而后遍历所有节点 v，如果节点 v 未被访问且从起点经过节点 u 到达节点 v 的距离小于当前记录的 dist[v] 距离，同时存在一条边，则更新 dist[v]和 prev[v]，使得 dist[v]记录更短的距离，同时将 prev[v] 设置为节点 u。

```
for(int i = 0; i < 4; i++)
{
    dist[i] = INT8_MAX;
    prev[i] = -1;
    visited[i] = 0;
}

dist[0] = 0;

for(int i = 0; i < num; i++)
{
    int u = min_dist(dist, visited,num);
    visited[u] = 1;
    for (int v = 0; v < num; v++)
```

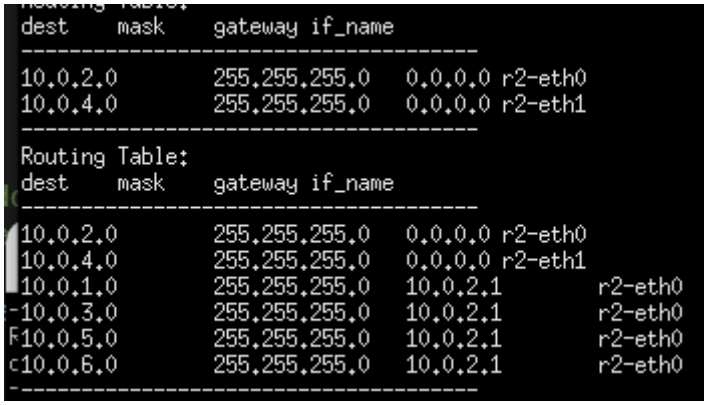
```
        {
            if (visited[v] == 0 && dist[u] + graph[u][v] < dist[v] && graph[u][v] > 0)
            {
                dist[v] = dist[u] + graph[u][v];
                prev[v] = u;
            }
        }
    }
```

2、结果



图表 6：h1 和 h2 路径

h1 节点对 h2 节点执行 traceroute 操作，可见路径查找正常。执行` link r2 r4 down `命令，取消 r2 和 r4 节点间的链路。继续等待约 40s，重新执行 h1 到 h2 的 traceroute 操作，可见路径发生改变且符合理论情况。

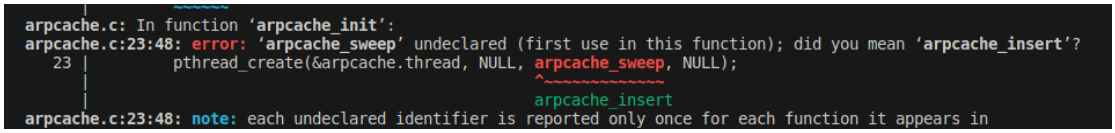


图表 7：路由器 r2 路由表

如图可见路由器 r2 在断开 r2 和 r4 的连接后路由表从只有两个局域网去向变为正常连接至所有局域网。

五、 遇到的问题

1、 有一个函数未声明



图表 8：函数未声明

2、 和上一个实验相同，编译代码时会出现很多 warning，以及似乎拓扑文件的一些书写格式和 python3 不兼容

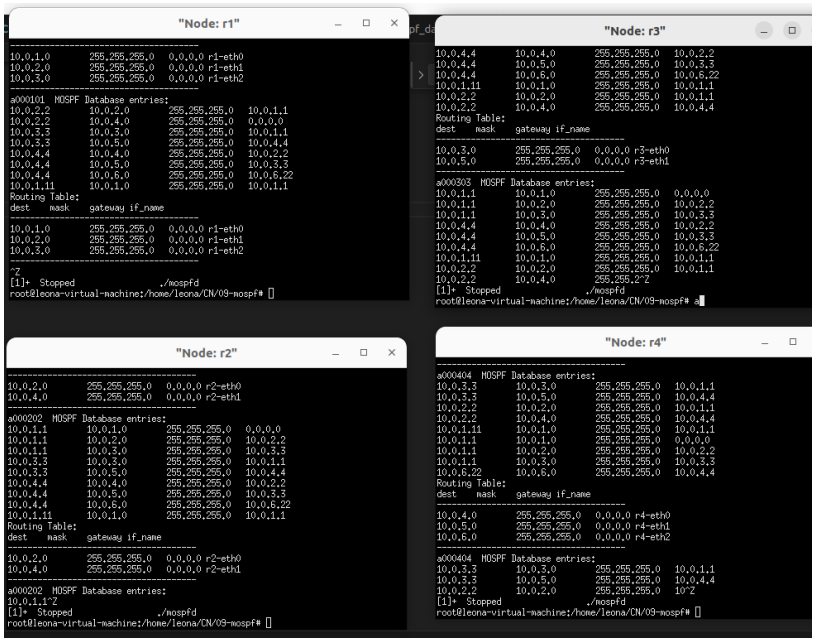
```
icmp.c:37:17: note: include '<string.h>' or provide a declaration of 'memset'
icmp.c:37:17: warning: incompatible implicit declaration of built-in function 'memset' [-Wbuiltin-declaration-mismatch]
icmp.c:37:17: note: include '<string.h>' or provide a declaration of 'memset'
icmp.c:38:17: warning: incompatible implicit declaration of built-in function 'memcpy' [-Wbuiltin-declaration-mismatch]
   38 |                 memcpy(send_pkt + ETHER_HDR_SIZE + IP_HDR_SIZE(iph) + 4 + 4, in_ip_hdr, IP_HDR_SIZE(in_ip_hdr) + 8);
      |                 ^~~~~~
```

图表 9：warning

```
leona@leona-virtual-machine:~/CN/09-mospf$ sudo python3 topo.py
[sudo] password for leona:
  File "/home/leona/CN/09-mospf/topo.py", line 18
    print '%s should be set executable by using `chmod +x $script_name`' % (fname)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?
```

图表 10：部分语法错误

3、 发现链路状态数据库可以正常生成但是 ping 不对

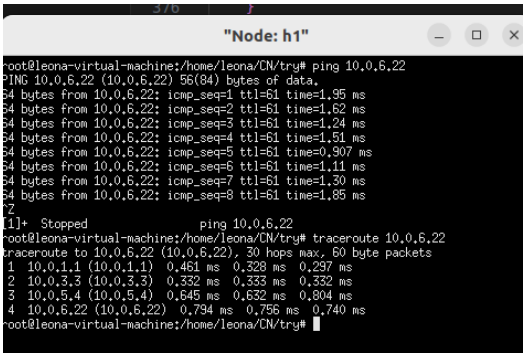


图表 11：链路状态数据库和路由表

链路状态数据库没有问题，但是和数据库一同输出 rtable 时，rtable 没有成功生成，只有端口所对网络的信息。因此考虑时路由生成函数 generate_rtable_thread，在该函数中输出 rtable，结果发现各路由器端点没有输出，因此该函数可能根本没有运行，查找该函数相关语句，发现确实没有运行。检查 mospf_run 函数，发现没有创建线程运行该函数。

```
pthread_create(&rtable, NULL, generate_rtable_thread, NULL);
```

添加上后，成功运行。



图表 12：输出正常