网络路由实验报告

张翔雨 2018K8009929035

一、实验题目：网络路由实验

二、实验内容

- 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
    - 运行网络拓扑(topo.py)
    - 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
    - 运行./mospfd，使得各个节点生成一致的链路状态数据库
- 基于实验一，实现路由器计算路由表项的相关操作
    - 运行网络拓扑(topo.py)
    - 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
    - 运行./mospfd，使得各个节点生成一致的链路状态数据库
    - 等待一段时间后，每个节点生成完整的路由表项
    - 在节点h1上ping/traceroute节点h2
    - 关掉某节点或链路，等一段时间后，再次用h1去traceroute节点h2

三、实验过程

1.实现各个节点生成一致的链路状态数据库

（1）实现hello消息和LSU消息的发送

hello消息的发送：`void *sending_mospf_hello_thread(void *param)`

该功能由一个线程单独实现，实现每个节点周期性（5秒）宣告自己的存在，消息中包括自己的节点ID，端口的子网掩码等消息。

```
void *sending_mospf_hello_thread(void *param)
{
    while(1)
    {
        pthread_mutex_lock(&mospf_lock);
```

```
        iface_info_t * iface = NULL;
        list_for_each_entry (iface, &instance->iface_list, list)
        {
            int size = ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + MOSPF_HDR_SIZE +
MOSPF_HELLO_SIZE;
            char * packet = (char*)malloc(size * sizeof(char));
            memset(packet, 0, size);

            struct ether_header *eh = (struct ether_header *)packet;
            struct iphdr *ip_hdr = packet_to_ip_hdr(packet);
            struct mospf_hdr *mospf_header = (struct mospf_hdr *)(packet +
ETHER_HDR_SIZE + IP_BASE_HDR_SIZE);
            struct mospf_hello *hello = (struct mospf_hello *)((char
*)mospf_header + MOSPF_HDR_SIZE);

            //ether head
            eh->ether_type = htons(ETH_P_IP);
            memcpy(eh->ether_shost, iface->mac, ETH_ALEN);
            eh->ether_dhost[0] = 0x01;
            eh->ether_dhost[2] = 0x5e;
            eh->ether_dhost[5] = 0x05;//目的IP地址为224.0.0.5，目的MAC地址为
01:00:5E:00:00:05
            // ip head
            ip_init_hdr(ip_hdr,iface->ip,MOSPF_ALLSPFRouters,IP_BASE_HDR_SIZE +
MOSPF_HDR_SIZE + MOSPF_HELLO_SIZE,IPPROTO_MOSPF);
            //mospf head
            mospf_init_hdr(mospf_header,MOSPF_TYPE_HELLO,MOSPF_HDR_SIZE +
MOSPF_HELLO_SIZE,instance->router_id,instance->area_id);
            hello
            mospf_init_hello(hello, iface->mask);

            mospf_header->checksum = mospf_checksum(mospf_header);
            iface_send_packet(iface, packet,size);
        }
        pthread_mutex_unlock(&mospf_lock);
        sleep(MOSPF_DEFAULT_HELLOINT);
    }
    return NULL;
}
```

LSU消息的发送：`void sending_mospf_lsu()`

该功能由于被多个线程调用，因此用单独函数实现，并在LSU消息发送线程中每30s调用一次，实现链路状态的扩散。

信息包括：该LSU的节点标识（RID，一般为路由器第1个端口的IP地址）；该节点的相邻节点列表（网络地址和对端节点标识）；序列号，用于区分不同的链路状态更新

```
void sending_mospf_lsu()
{
```

```
    iface_info_t * iface = NULL;
    int nbr = 0;
    //计算得到邻居数目
    list_for_each_entry (iface, &instance->iface_list, list)
    {
        if (iface->num_nbr == 0)
            nbr ++;
        else
            nbr += iface->num_nbr;
    }

    struct mospf_lsa *lsa_array = (struct mospf_lsa*)malloc(nbr * MOSPF_LSA_SIZE);
    memset(lsa_array,0,nbr * MOSPF_LSA_SIZE);

    int i = 0;

    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++;
            }
        }
    }

    instance->sequence_num++;

    int size = ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + MOSPF_HDR_SIZE + MOSPF_LSU_SIZE
+ nbr * MOSPF_LSA_SIZE;

    iface = NULL;
    list_for_each_entry (iface, &instance->iface_list, list)
    {
        mospf_nbr_t * ptr = NULL;
        list_for_each_entry (ptr, &iface->nbr_list, list)
        {
            char * packet = (char*)malloc(size);
            memset(packet,0,size);
            struct ether_header *eh = (struct ether_header *)packet;
```

```
            struct iphdr *ip_hdr = packet_to_ip_hdr(packet);
            struct mospf_hdr *mospf_header = (struct mospf_hdr *)(packet +
ETHER_HDR_SIZE + IP_BASE_HDR_SIZE);
            struct mospf_lsu *lsu = (struct mospf_lsu *)((char *)mospf_header +
MOSPF_HDR_SIZE);
            struct mospf_lsa *lsa = (struct mospf_lsa *)((char *)lsu +
MOSPF_LSU_SIZE);

            eh->ether_type = htons(ETH_P_IP);
            memcpy(eh->ether_shost, iface->mac, ETH_ALEN);

            ip_init_hdr(ip_hdr,iface->ip,ptr->nbr_ip, size -
ETHER_HDR_SIZE,IPPROTO_MOSPF);

            mospf_init_hdr(mospf_header,MOSPF_TYPE_LSU,MOSPF_HDR_SIZE +
MOSPF_LSU_SIZE + nbr * MOSPF_LSA_SIZE,instance->router_id,instance->area_id);

            mospf_init_lsu(lsu, nbr);
            memcpy(lsa, lsa_array, nbr * MOSPF_LSA_SIZE);

            mospf_header->checksum = mospf_checksum(mospf_header);
            ip_send_packet(packet, size);
        }
    }
}
```

（2）实现hello消息和LSU消息的处理

实现hello消息的处理：`void handle_mospf_hello(iface_info_t *iface, const char *packet, int *len*)`

当节点收到hello消息时，从包中得到相关的信息。如果发送该消息的节点不在邻居列表中，添加至邻居列表；如果已存在，更新其达到时间。

```
void handle_mospf_hello(iface_info_t *iface, const char *packet, int len)
{
    struct iphdr *ip_hdr = packet_to_ip_hdr(packet);
    struct mospf_hdr *mospf_header = (struct mospf_hdr *)(packet + ETHER_HDR_SIZE
+ IP_BASE_HDR_SIZE);
    struct mospf_hello *hello = (struct mospf_hello *)((char *)mospf_header +
MOSPF_HDR_SIZE);
    pthread_mutex_lock(&mospf_lock);
    int find = 0;
    mospf_nbr_t *nbr = NULL;
    list_for_each_entry(nbr, &iface->nbr_list, list)
    {
        if (nbr->nbr_id == ntohl(mospf_header->rid))
        {
            find = 1;
            nbr->alive = 0;
```

```
            break;
        }
    }
    if (!find)
    {
        mospf_nbr_t *new_nbr = (mospf_nbr_t*)malloc(sizeof(mospf_nbr_t));
        new_nbr->nbr_id = ntohl(mospf_header->rid);
        new_nbr->nbr_ip = ntohl(ip_hdr->saddr);
        new_nbr->nbr_mask = ntohl(hello->mask);
        new_nbr->alive = 0;
        list_add_tail(&new_nbr->list, &iface->nbr_list);
        iface->num_nbr++;
        sending_mospf_lsu();
    }
    pthread_mutex_unlock(&mospf_lock);
}
```

实现LSU消息的处理：`void handle_mospf_lsu(iface_info_t *iface, char *packet, int len)`

当收到链路状态信息时，如果之前未收到该节点的链路状态信息，或者该信息的序列号更大，则更新链路状态数据库，并TTL减1，如果TTL值大于0，则向除该端口以外的端口转发该消息。

```
void handle_mospf_lsu(iface_info_t *iface, char *packet, int len)
{
    struct iphdr *ip_hdr = packet_to_ip_hdr(packet);
    struct mospf_hdr *mospf_header = (struct mospf_hdr *)(packet + ETHER_HDR_SIZE
+ IP_BASE_HDR_SIZE);
    struct mospf_lsu *lsu = (struct mospf_lsu *)((char *)mospf_header +
MOSPF_HDR_SIZE);
    struct mospf_lsa *lsa = (struct mospf_lsa *)((char *)lsu + MOSPF_LSU_SIZE);

    pthread_mutex_lock(&mospf_lock);
    int find = 0;
    int updated = 0;

    mospf_db_entry_t * db = NULL;
    //在已有信息中查找
    list_for_each_entry (db, &mospf_db, list)
    {
        if (db->rid == ntohl(mospf_header->rid))
        {
            find = 1;
            if (db->seq < ntohs(lsu->seq))
            {
                updated = 1;
                db->seq = ntohs(lsu->seq);
                db->nadv = ntohl(lsu->nadv);
                db->alive = 0;
                for (int i = 0; i < db->nadv; i++)
```

```
                {
                    db->array[i].network = ntohl(lsa[i].network);
                    db->array[i].mask = ntohl(lsa[i].mask);
                    db->array[i].rid = ntohl(lsa[i].rid);
                }
            }
        }
    }
    //若未找到，生成新的数据加入
    if (!find)
    {
        updated = 1;
        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);
    }

    pthread_mutex_unlock(&mospf_lock);
    //如果链路状态数据发生了更新，进行ttl-1，如果结果大于0则转发。
    if(updated)
    {
        lsu->ttl -- ;
        if (lsu->ttl > 0)
        {
            iface_info_t * iface = NULL;
            list_for_each_entry (iface, &instance->iface_list, list)
            {
                mospf_nbr_t *nbr = NULL;
                list_for_each_entry(nbr, &iface->nbr_list, list)
                {
                    if (nbr->nbr_id != ntohl(mospf_header->rid))
                    {
                        char *forward = (char *)malloc(len);
                        struct iphdr * iph = packet_to_ip_hdr(forward);
                        struct mospf_hdr * mospfh = (struct mospf_hdr *)(forward +
ETHER_HDR_SIZE + IP_BASE_HDR_SIZE);

                        memcpy(forward, packet, len);

                        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);
                    }
                }
            }
        }
        //打印数据库信息
        fprintf(stdout, "%x  ",instance->router_id);
        fprintf(stdout, "MOSPF Database entries:\n");
        list_for_each_entry(db, &mospf_db, list)
        {
            for (int i = 0; i < db->nadv; i++)
                fprintf(stdout, IP_FMT"\t"IP_FMT"\t"IP_FMT"\t"IP_FMT"\n",
HOST_IP_FMT_STR(db->rid), HOST_IP_FMT_STR(db->array[i].network),
HOST_IP_FMT_STR(db->array[i].mask), HOST_IP_FMT_STR(db->array[i].rid));
        }

    }
}
```

2.实现路由器计算路由表项的相关操作

（1）将链路状态数据库抽象成图拓扑

这里定义存放路由器信息的数组和，存放图拓扑信息的二维数组，根据当前链路状态数据库生成图拓扑。

```
void init_graph()
{
    memset(graph, INT8_MAX -1, sizeof(graph));
    mospf_db_entry_t *db = NULL;
    router_list[0] = instance->router_id;
    num = 1;

    list_for_each_entry(db, &mospf_db, list)
    {
        router_list[num] = db->rid;//将每个路由表号存在路由器列表中
        num++;
    }

    db = NULL;
    list_for_each_entry(db, &mospf_db, list)
    {
        int u = locate_entry(db->rid);
        for(int i = 0; i < db->nadv; i ++ )
        {
```

```
            if(db->array[i].rid)
            {
                int v = locate_entry(db->array[i].rid);//找到路由器在列表里对应的ID
                graph[u][v] = graph[v][u] = 1;//在拓扑图中标志两个节点连通
            }
        }
    }
}
```

（2）使用Dijkstra算法计算每个节点的最短路径和上一跳节点

创建dist[]和prev[]数组

```
void Dijkstra(int prev[], int dist[])
{
    int visited[4];
    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);//在未访问的节点中，选取离已访问节点最近的那
个
        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;
            }
        }
    }
}
```

（3）根据生成的prev和dist数组重新生成路由表

这里使用一个单独的线程来重新计算路由表，每隔1s更新一次，效率比较高。

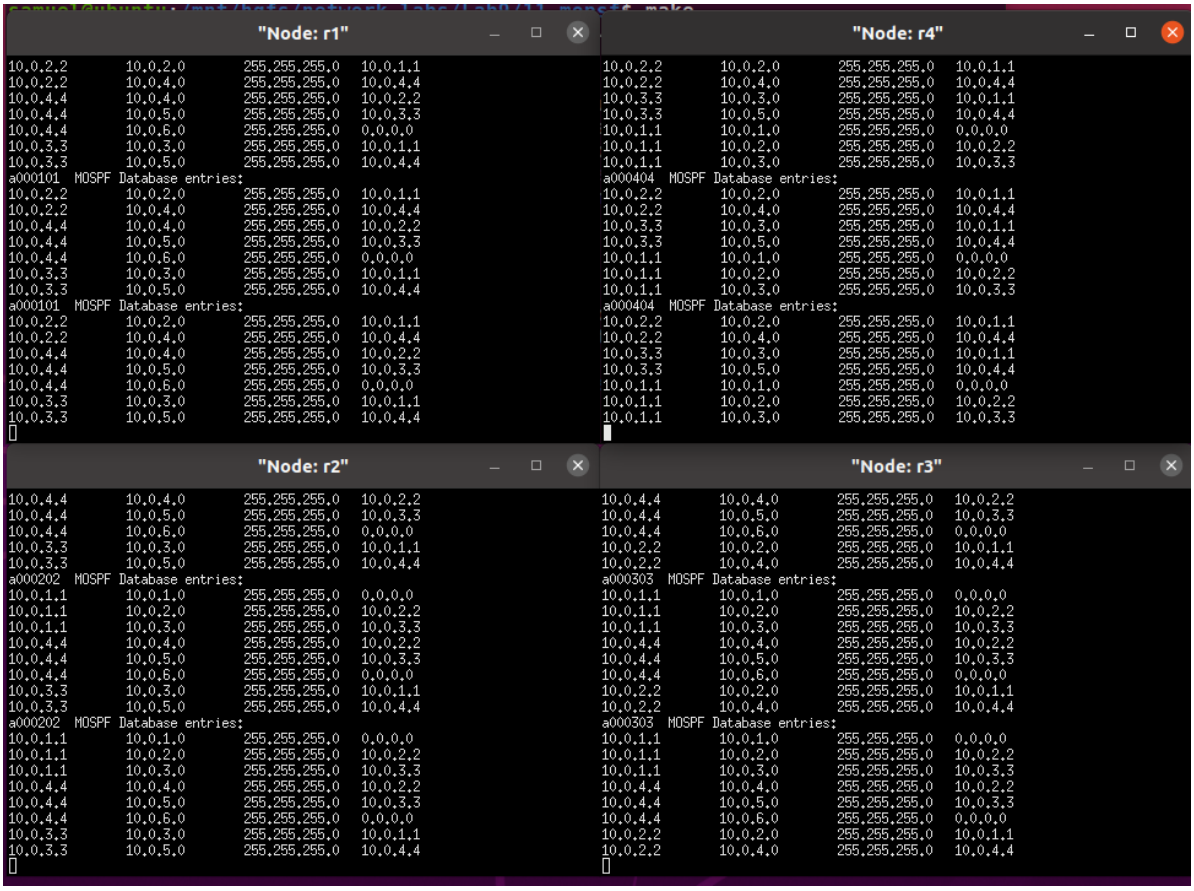
```
void *generate_rtable_thread(void *param)
{
    while(1) {
```

```
        sleep(5);
        int prev[4];
        int dist[4];
        init_graph();
        Dijkstra(prev, dist);
        int visited[4] = {0};
        visited[0] = 1;

        rt_entry_t *rt_entry, *rt_entry_next;
        list_for_each_entry_safe (rt_entry, rt_entry_next, &rtable, list) //删除路由表中内核加载的节点
        {
            if(rt_entry->gw)
                remove_rt_entry(rt_entry);
        }
        for (int i = 0; i < num; i ++ )
        {
            int t = -1;
            for(int j = 0; j < num; j ++ )
            {
                if(!visited[j])
                {
                    if(t == -1 || dist[j] < dist[t])//最短路径
                        t = j;
                }
            }
            visited[t] = 1;
            mospf_db_entry_t *db;
            list_for_each_entry(db, &mospf_db, list)
            {
                if(db->rid == router_list[t])
                {
                    int next_hop;
                    while(prev[t] != 0)
                        t = prev[t];
                    next_hop = t;//找到下一跳节点
                    iface_info_t *iface;
                    u32 gw;
                    int find = 0;

                    list_for_each_entry (iface, &instance->iface_list, list)
                    {
                        mospf_nbr_t *nbr;
                        list_for_each_entry (nbr, &iface->nbr_list, list)
                        {
                            if(nbr->nbr_id == router_list[next_hop])
                            {
                                find = 1;
                                gw = nbr->nbr_ip;
                                break;
                            }
                        }
```

```
                        if(find)
                            break;
                    }
                    if(!find)
                        break;
                    for(int i = 0; i < db->nadv; i ++)
                    {
                        rt_entry_t *entry = NULL;
                        find = 0;
                        list_for_each_entry(entry, &rtable, list)
                        {
                            if(entry->dest == db->array[i].network && entry->mask
== db->array[i].mask)
                            {//找到相印需要更新的信息
                                find = 1;
                                break;
                            }
                        }
                        if(!find)
                        {
                            rt_entry_t *new = new_rt_entry(db->array[i].network,
db->array[i].mask, gw, iface);//将新信息加到路由表中
                            add_rt_entry(new);
                        }
                    }
                }
            }
        }
        print_rtable();//打印路由表
    }
}
```

四、实验结果

- **1.实验内容1：生成一致性链路状态数据库**



可以看到生成了正确的一致性链路状态数据库。

• 2.实验内容2：计算生成路由表项

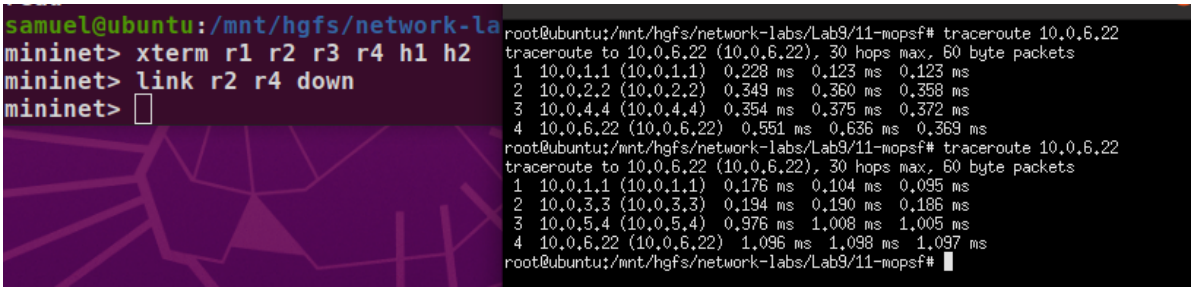
实验中可以观察到路由器节点生成了例如下图r2所示的路由表。

```
Routing Table:
dest    mask    gateway if_name
-------------------------------------
10.0.2.0        255.255.255.0   0.0.0.0 r2-eth0
10.0.4.0        255.255.255.0   0.0.0.0 r2-eth1
10.0.1.0        255.255.255.0   10.0.2.1        r2-eth0
10.0.3.0        255.255.255.0   10.0.2.1        r2-eth0
10.0.5.0        255.255.255.0   10.0.4.4        r2-eth1
10.0.6.0        255.255.255.0   10.0.4.4        r2-eth1
-------------------------------------
```

在mininet中执行 `link r2 r4 down` ，可以观察到路由表的变化，以r2为例：

```
Routing Table:
dest    mask    gateway if_name
-------------------------------------
10.0.2.0        255.255.255.0   0.0.0.0 r2-eth0
10.0.4.0        255.255.255.0   0.0.0.0 r2-eth1
10.0.1.0        255.255.255.0   10.0.2.1        r2-eth0
10.0.3.0        255.255.255.0   10.0.2.1        r2-eth0
10.0.5.0        255.255.255.0   10.0.2.1        r2-eth0
10.0.6.0        255.255.255.0   10.0.2.1        r2-eth0
-------------------------------------
```

两次 `h1 traceroute h2` 的结果如下



路径输出正常。

五、思考题

1.在构建一致性链路状态数据库中，为什么邻居发现使用组播(Multicast)机制，链路状态扩散用单播(Unicast)机制?

邻居发现是为了节点周期性的发现是否有新的邻居，通过没有特定目的地址的广播能够发现自己的新邻居，并加入邻居列表。若使用单播则必须知道目的地址，无法分现新加入的邻居。

链路状态扩散是要将当前的链路信息发送给邻居节点，需要精准地送到目的地址，而无需送给其他非邻居节点，占用带宽，因此使用单播机制。

2.该实验的路由收敛时间大约为20-30秒，网络规模增大时收敛时间会进一步增加，如何改进路由算法的可扩展性?

- 在实验中发现产生了大量的广播包，可以改进广播算法减少链路中的冗余广播包，来提高链路的有效利用率，以提升收敛速度。
- 洪泛算法导致了链路中每个邻节点都会转发收到的报文，当目的节点收到重复报文时，会丢弃掉当前报文，可以实现邻节点有概率转发或者选择转发以提高链路利用率

3.路由查找的时间尺度为~ns，路由更新的时间尺度为~10s，如何设计路由查找更新数据结构，使得更新对查找的影响尽可能小?

- 简单的控制可以实现为在路由表数据结构中加入线程锁，查找和更新的线程访问路由表时都需要申请锁，保证互斥访问
- 也可以在路由表中实现一个查询队列，所有的路由查找请求被挂在查询队列中顺序执行，每次路由表更新时，会遍历查询队列，若为当前正在更新的信息，则直接返回查询结果，在更新过程中遍历路由表时，若找到对应查询队列中的项，也会返回查询结果。

六、实验总结

本次实验代码量比较多，同时对数据结构中的图算法进行了复习，实验内容1比较简单但内容2耗费了较多时间。通过实验代码的书写，我也逐渐对路由表的计算生成过程有了更深的了解。