Mospf 实验设计报告

中国科学院大学

页

2022 年 5 月 6 日

一、网络路由实验

1. 实验内容

1）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。路由器需要完成的操作：

①周期性（hello-interval ：5 秒）宣告自己的存在；

②节点收到 mOSPF Hello 消息后，维护邻居列表;

③周期性老化邻居列表；

④创建链路状态数据包 LSU 并发送，向邻居结点通告自己的链路状态信息；

⑤处理 LSU 包，生成链路状态数据库记录网络拓扑。

2）基于（1），实现路由器计算路由表项的相关操作：

①根据网络拓扑结构计算到网络中其他结点的最短路径，生成路由表；

②当链路状态发生变化时，能重新生成当前网络的路由表。

在给定网络拓扑下运行 traceroute 测试，断开一条链路（r2-r4）后，经过一段时间，再次测试。

2. 实验流程

（1） 将 mOSPF 解析脚本 mospf.lua 按照讲义方法加入 Wireshark 插件

（2） 编写实验代码，完成文件 mospf_daemon.c 中 sending_mospf_hello_thread、checking_nbr_thread 、 checking_database_thread 、 handle_mospf_hello 、 和 sending_mospf_lsu_thread 等函数

（3）测试并检验路由器生成的数据库信息：

运行网络拓扑(topo.py)，拓扑结构如图 1 所示。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，在路由节点上运行./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(r2-r4)，等一段时间后，再次用 h1 去

traceroute 节点 h2。

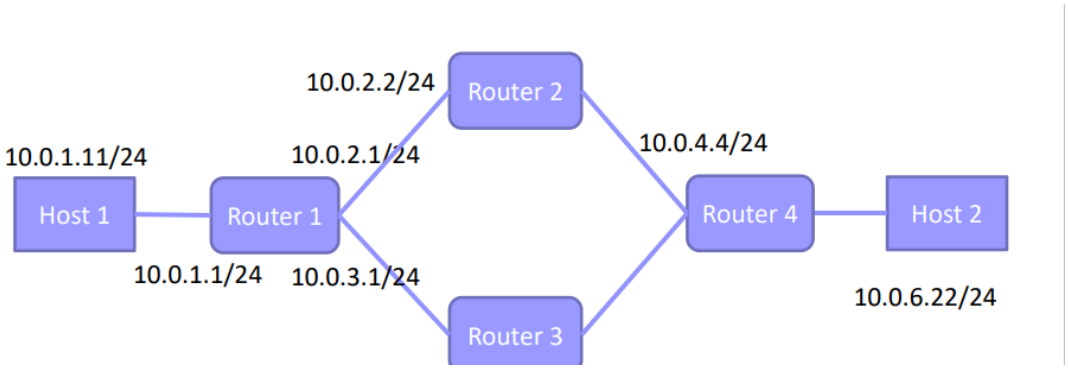

图 1

在 topo.py 中添加测试部分代码图下所示：

```
    for r in (r1, r2, r3, r4):
        r.cmd('./mospfd > %s-output.txt 2>&1 &' % r)
        # r.cmd('./mospfd-reference > %s-output.txt 2>&1 &' % r)

    sleep(40)
    print(h1.cmd('traceroute 10.0.6.22'))

    net.delLinkBetween(r2,r4)
    sleep(40)
    print(h1.cmd('traceroute 10.0.6.22'))
    sleep(3)
    for r in (r1, r2, r3, r4):
        r.cmd('pkill -SIGTERM mospfd')
        # r.cmd('pkill -SIGTERM mospfd-reference')
```

3. 实验设计

(1) 总体设计

在上一次实验中我们实现了静态路由协议，静态路由配置方便，对系统要求低，适用于拓扑结构简单并且稳定的小型网络。但它的缺点页非常明显，不能自动适应网络拓扑的变化，需要人工干预。相比之下动态路由协议有自己的路由算法，消耗一定的网络资源和系统资源使路由器能够自动适应网络拓扑的变化，适用于具有一定数量设备的网络。

本次实验基于 MOSPF（Multicast Open Shortest Path First）协议，即组播开放式最短路径优先协议。OSPF 是一种链路状态协议。可以将链路视为路由器的接口。链路状态是对接口及接口与相邻路由器的关系的描述。接口的信息包括接口的 IP 地址、掩码、所连接的网络的类型、连接的邻居等。所有这些链路状态的集合形成链路状态数据库。

OSPF工作原理可分为三步：

1. 邻居发现：路由器周期宣告自己的存在，不同路由之间之间建立邻居关系；
2. 链路状态的扩散和更新：每台路由器产生并向邻居洪泛链路状态信息，同时接收并

处理来自其他路由器的链路状态信息，最终达到网络内路由得到相同网络拓扑的状态；

3. 计算最优路由：根据 Dijkstra 算法计算本节点到网络中每个结点的最短路径，然后根据计算结果更新路由表。

MOSPF 是通过在 OSPF 链接状态通告中包含组播信息而工作的，MOSPF 与 OSPFv2 的区别在于：

1. OSPFv2 的 protocol number 为 89，而 MOSPF 为 90；
2. MOSPF 对数据包格式进行了适当简化；
3. OSPFv2 基于可靠洪泛：收到 LSU 数据包后需要回复 ACK；
4. OSPFv2 有更多的消息类型。例如，链路状态数据库 Summary；
5. OSPFv2 有安全认证机制（鉴别）；

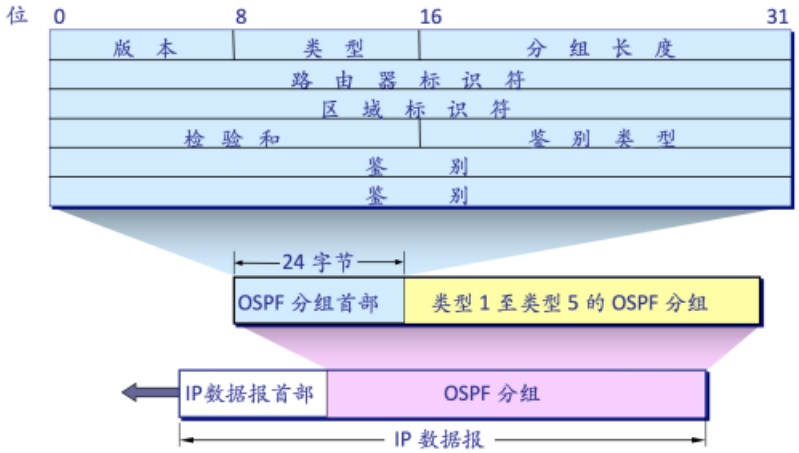

图 2

MOSPF 数据包格式和图 2 中 OSPFv2 数据包格式相同。本次实验中，需要用到类型 1(HELLO)和类型 4(LSU)两种分组。

HELLO 分组：

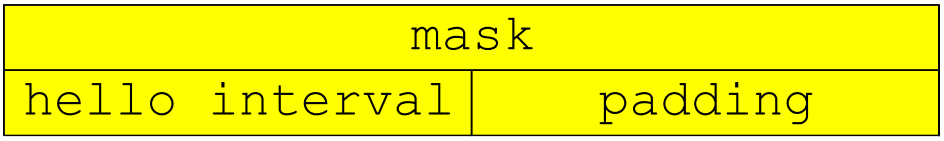

图 3

①mask：生成本消息的端口的掩码。

hello interval：两次发送 hello 消息之间间隔的时间。

padding：填充保持对齐

②LSU 分组：

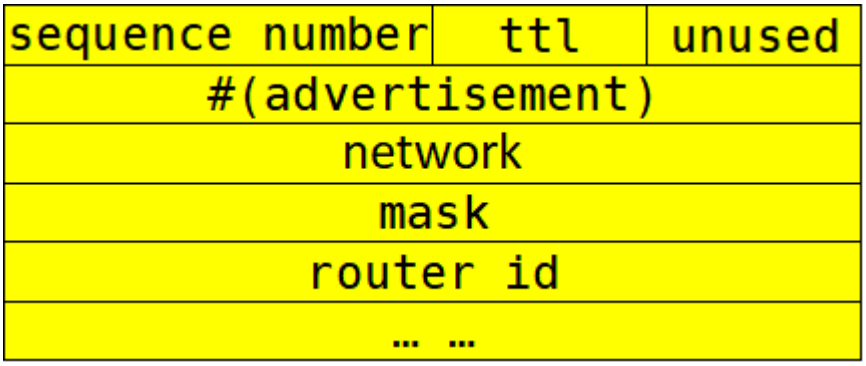

图 4

sequence number： LSU 的序列号。

ttl ：剩余生存时间；

#(advertisement：邻居的个数；

network：邻居所在的网段；

mask：邻居所在网段的掩码；

router id ：邻居路由器 ID 。

(2)实现代码（mospf_daemon.c）

在初始化以后，mospf 会启动四个不同的线程 sending_mospf_hello_thread：周期地发送 hello 宣告自己的存在，sending_mospf_lsu_thread：周期性发送 lsu 数据包，checking_nbr_thread：周期检查邻居列表中老化的节点，checking_database_thread：周期性的检查数据库中老化的节点。同时在路由器接收到数据包时，如果收到的协议格式为 IPPROTO_MOSPF 将会调用 handle_mospf_packet 函数进行处理，在检查版本号，校验和后又根据数据包时 hello 和 lsu 分别调用 handle_mospf_hello 函数和 handle_mospf_lsu 函数进行处理。下面简要上述调用函数的处理流程

①void *sending_mospf_hello_thread(void *param)

节点每间隔 HELLOINT(5s)按图 2 和图 3 的格式填充 hello 包，组播本节点的信息向外界宣告自己的存在。首先配置以太网首部和 ip 首部，其中组播目的 MAC 地址为 01:00:5E:00:00:05，目的 IP 地址为 224.0.0.5，然后配置 mospf 首部，计算校验和之后即可发送数据包。

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

            mospf_init_hdr(mospf_header, MOSPF_TYPE_HELLO,
                           MOSPF_HDR_SIZE + MOSPF_HELLO_SIZE,
```

```
                        instance->router_id, 0);

            mospf_init_hello(hello, iface->mask);
            mospf_header->checksum = mospf_checksum(mospf_header);
```

②sending_mospf_lsu_thread

每隔一个 MOSPF_DEFAULT_LSUINT (30s)，按图 4 的格式填充 lsu 包，向所有邻居节点发送自己的链路状态信息。主要流程为统计自己的相邻节点，按照链路状态配置 lsu 数据包，向所有邻居组播 lsu，当端口没有相邻路由器时，也要表达该网络，邻居节点 ID 为 0。

④void *checking_nbr_thread(void *param)

如果在 MOSPF_DATABASE_TIMEOUT 时间内没有更新链路状态，需要老化删除数据库中对应的结点。

```
        list_for_each_entry_safe(entry, q, &mospf_db, list)
        {
            if (entry->alive++ >= MOSPF_DATABASE_TIMEOUT)
            {
                list_delete_entry(&entry->list);
                free(entry);
            }
        }
```

⑤void handle_mospf_hello(iface_info_t *iface, const char *packet, int len)

对于收到的 hello 包，获得其路由器 ID 、 IP 、 MASK 等节点信息。查找邻居节点链表，若存在此节点，则更新其生存时间，否则新增一个链表项存储该节点信息，并向每个邻居节点发送 lsu 包组播链路发生的改变。访问邻居节点链表时，需要上锁，以和清除过期邻居节点的线程互斥。

```
    list_for_each_entry(nbr, &iface->nbr_list, list)
    {
        if (nbr->nbr_id == rid)
        {
            nbr->alive = 0;
            pthread_mutex_unlock(&mospf_lock);
            return;
        }
    }
    nbr = malloc(sizeof(mospf_nbr_t));
    nbr->nbr_id = rid;
    nbr->nbr_ip = ntohl(ihr->saddr);
    nbr->alive = 0;
    nbr->nbr_mask = ntohl(hello->mask);
```

```
    list_add_tail(&nbr->list, &iface->nbr_list);
    iface->num_nbr++;
    pthread_mutex_unlock(&mospf_lock);
    send_mospf_lsu();
```

⑥void handle_mospf_lsu(iface_info_t *iface, char *packet, int len)

读取 lsu 包链路状态信息。遍历链路状态数据库，若找到相应的节点且其信息较旧(即收到的 LSU 序列号更大)则更新信息，否则新建一个节点存储其信息。若链路发生了变化.即更新了信息或新建了节点，则需要更新路由表，然后向所有邻居节点转发该 lsu 包。访问链路状态数据库需要上锁以实现互斥。

```
    list_for_each_entry_safe(entry, q, &mospf_db, list)
    {
        if (entry->rid == rid)
        {
            if (entry->seq >= seq)
            {
                pthread_mutex_unlock(&db_lock);
                return;
            }
            found = 1;
            free(entry->array);
            break;
        }
    }
    if (!found)
    {
        entry = (mospf_db_entry_t *)malloc(sizeof(mospf_db_entry_t));
        list_add_tail(&entry->list, &mospf_db);
    }
    entry->rid = rid;
    entry->seq = seq;
    entry->nadv = nadv;
    entry->alive = 0;
    entry->array = (struct mospf_lsa *)malloc(nadv * sizeof(struct
mospf_lsa));
    struct mospf_lsa *lsa = (struct mospf_lsa *)(lsu + 1);
    for (int i = 0; i < nadv; i++)
    {
        entry->array[i].rid = ntohl(lsa[i].rid);
        entry->array[i].network = ntohl(lsa[i].network);
        entry->array[i].mask = ntohl(lsa[i].mask);
    }
    pthread_mutex_unlock(&db_lock);
    update_rtable();
```

```
    if (--lsu->ttl > 0)
    {
        iface_info_t *iface_pos = NULL;
        list_for_each_entry(iface_pos, &instance->iface_list, list)
        {
            mospf_nbr_t *nbr = NULL;
            list_for_each_entry(nbr, &iface_pos->nbr_list, list)
            {
                if (nbr->nbr_id == ntohl(mhr->rid))
                    continue;
                char *forwarding_packet = (char *)malloc(len);
                memcpy(forwarding_packet, packet, len);

                struct iphdr *iph = packet_to_ip_hdr(forwarding_packet);
                iph->saddr = htonl(iface_pos->ip);
                iph->daddr = htonl(nbr->nbr_ip);

                struct mospf_hdr *mhr = (struct mospf_hdr *)((char *)iph +
IP_HDR_SIZE(iph));
                mhr->checksum = mospf_checksum(mhr);
                iph->checksum = ip_checksum(iph);

                ip_send_packet(forwarding_packet, len);
            }
        }
    }
```

⑦update_rtable()

当链路状态数据库发生变化后，需要更新路由表，更新路由表需要将路由表转换为图，然后利用 Dijkstra 算法计算出本节点到达其他节点的距离和路线，然后按照距离和路线更新路由表，更新路由表时如果源节点和目标节点不是相邻的，则需要找到源节点到目标节点中需要经过的网关，然后才能更新路由表.

```
void update_rtable()
{
    init_graph();
    Dijkstra(prev, dist);
    update_router(prev, dist);
}
```

```
void Dijkstra(int prev[], int dist[])
{
    int visited[ROUTER_NUM];
```

```
    for (int i = 0; i < ROUTER_NUM; i++)
    {
        dist[i] = INT8_MAX;
        prev[i] = FALSE;
        visited[i] = 0;
    }

    dist[0] = 0;

    for (int i = 0; i < num; i++)
    {
        //在没有访问的节点中选择离源节点最近的那个
        int u = min_dist(dist, visited, num);
        visited[u] = 1;
        //更新通过 u 后其他节点到源节点的距离
        for (int v = 0; v < num; v++)
        {
            if (visited[v] == 0 && dist[u] + graph[u][v] < dist[v])
            {
                dist[v] = dist[u] + graph[u][v];
                prev[v] = u;
            }
        }
    }
}
```

4. 实验结果与分析

(1) 测试并检验路由器生成的数据库信息:

r2:

RID	Network	Mask	Nbr
10.0.1.1	10.0.1.0	255.255.255.0	0.0.0.0
10.0.1.1	10.0.2.0	255.255.255.0	10.0.2.2
10.0.1.1	10.0.3.0	255.255.255.0	10.0.3.3
10.0.4.4	10.0.4.0	255.255.255.0	10.0.2.2
10.0.4.4	10.0.5.0	255.255.255.0	10.0.3.3
10.0.4.4	10.0.6.0	255.255.255.0	0.0.0.0
10.0.3.3	10.0.3.0	255.255.255.0	10.0.1.1
10.0.3.3	10.0.5.0	255.255.255.0	10.0.4.4

```
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
r3:
RID                     Network                 Mask
Nbr
10.0.1.1        10.0.1.0        255.255.255.0   0.0.0.0
10.0.1.1        10.0.2.0        255.255.255.0   10.0.2.2
10.0.1.1        10.0.3.0        255.255.255.0   10.0.3.3
10.0.4.4        10.0.4.0        255.255.255.0   10.0.2.2
10.0.4.4        10.0.5.0        255.255.255.0   10.0.3.3
10.0.4.4        10.0.6.0        255.255.255.0   0.0.0.0
10.0.2.2        10.0.2.0        255.255.255.0   10.0.1.1
10.0.2.2        10.0.4.0        255.255.255.0   10.0.4.4
Routing Table:
dest    mask    gateway if_name
--------------------------------------
10.0.3.0        255.255.255.0   0.0.0.0 r3-eth0
10.0.5.0        255.255.255.0   0.0.0.0 r3-eth1
10.0.1.0        255.255.255.0   10.0.3.1        r3-eth0
10.0.2.0        255.255.255.0   10.0.3.1        r3-eth0
10.0.4.0        255.255.255.0   10.0.5.4        r3-eth1
10.0.6.0        255.255.255.0   10.0.5.4        r3-eth1
```

r1 ,r4 省略

数据库内容一致,路由表正常生成

(2)测试路由变动,删除 r2-r4 链路,能否重新生成路由表,结果如下图 5 所示

```
stu@stu:~/computer_net/07-mospf$ sudo python topo.py
traceroute to 10.0.6.22 (10.0.6.22), 30 hops max, 60 byte packets
 1  10.0.1.1 (10.0.1.1)  0.154 ms  0.059 ms  0.053 ms
 2  10.0.2.2 (10.0.2.2)  0.064 ms  0.054 ms  0.108 ms
 3  10.0.4.4 (10.0.4.4)  0.194 ms  0.152 ms  0.140 ms
 4  10.0.6.22 (10.0.6.22)  0.186 ms  0.152 ms  0.137 ms

traceroute to 10.0.6.22 (10.0.6.22), 30 hops max, 60 byte packets
 1  10.0.1.1 (10.0.1.1)  0.150 ms  0.100 ms  0.090 ms
 2  10.0.3.3 (10.0.3.3)  0.223 ms  0.220 ms  0.215 ms
 3  10.0.5.4 (10.0.5.4)  1.789 ms  1.794 ms  1.787 ms
 4  10.0.6.22 (10.0.6.22)  1.781 ms  1.774 ms  1.768 ms
```

图 5

5. 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？

答：

邻居发现时路由器还不知道邻居的信息，也就不知道其 IP 地址，无法使用单播，只能采用组播.把参与 mOSPF 的路由器设置相同的组播地址，而在发送链路状态的 LSU 消息时，路由器已经知道自己有哪些邻居，也知道其 IP 地址，因此可以使用单播。

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？

答：

可以应用使用划分区域的方法,每个区域有不同的 area id 链路状态信息的泛洪只在各区域内部进行，路由器不处理其他区域的链路状态信息，这样各区域可以分别快速收敛，最后各区域再交换获取其他区域信息来达到整个网络的收敛

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？

答：

本实验中每次更新路由表都要清理原来的非默认路由项后重新计算。在计算最短路径时时，计算时间过长,而此时查找被停止就十分的浪费.因此我们可以不停止查找,将当前要更新的结果先放在一个缓存里面,而路由查找继续正常运行,等到路由器计算更新完新的路由表,再停止路由查找,将缓存内的新路由表放进实际的路由表中。