# 实验报告

孔静 2014K8009929022

November 26, 2017

## Contents



## 1 实验题目

- 网络路由实验一

## 2 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
- 运行实验
  - 将 wireshark 文件夹下的 mospf.lua 放到 /.wireshark/plugins 目录
  - 运行网络拓扑 (topo/mospf_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
  - 关掉 r2 中的 mospfd 程序，网络拓扑发生变动，各个节点仍可以生成一致的链路状态数据库

# 3 实验流程

## 3.1 文件列表

```
lab10
├── homework.pdf
└── 10-mospf
    ├── mospf_daemon.c
    ├── include
    │   ├── mospf_database.h
    │   ├── mospf_proto.h
    │   └── ...
    └── ...
```

## 3.2 相关代码

- 邻居节点及数据库的老化操作

```
void *checking_database_thread(void *param){
  mospf_db_entry_t* db, *db_q;
  while(1){
    sleep(1);
    pthread_mutex_lock(&mospf_lock);
    list_for_each_entry_safe(db, db_q, &mospf_db, list){
      if(--db->alive == 0){
        list_delete_entry(&db->list);
        free(db->array);
        free(db);
      }
    }
    pthread_mutex_unlock(&mospf_lock);
  }
  return NULL;
}
void *checking_nbr_thread(void *param)
{
  // fprintf(stdout, "TODO: neighbor list timeout operation.\n");
    iface_info_t *iface;
    mospf_nbr_t *nbr, *nbr_q;
    mospf_db_entry_t *db;
    while (1)
    {
        sleep(1);
        pthread_mutex_lock(&mospf_lock);
        list_for_each_entry(iface, &(instance->iface_list), list){
            list_for_each_entry_safe(nbr, nbr_q, &(iface->nbr_list), list){
                if(--nbr->alive == 0){
                    list_delete_entry(&(nbr->list));
                    iface->num_nbr -= 1;
                    broadcast_mospf_lsu();
                    list_for_each_entry(db, &mospf_db, list){
                        if (db->rid == nbr->nbr_id){
                            free(db->array);
                            list_delete_entry(&db->list);
                            free(db);
                            break;
                        }
                    }
                    free(nbr);
                }
            }
        }
        pthread_mutex_unlock(&mospf_lock);
    }
    return NULL;
```

```
}
```

- 定期发送 hello 消息

```
void *sending_mospf_hello_thread(void *param)
{
  // fprintf(stdout, "TODO: send mOSPF Hello message periodically.\n");
    int length = LSU_OFFSET + MOSPF_HELLO_SIZE;
    char *packet;
    struct ether_header *eh;
    struct iphdr *ip;
    struct mospf_hdr *hdr;
    struct mospf_hello *hello;
    iface_info_t *iface;
    while (1)
    {
        sleep(MOSPF_DEFAULT_HELLOINT);
        pthread_mutex_lock(&mospf_lock);
        list_for_each_entry(iface, &(instance->iface_list), list){
            packet = (char *)malloc(length);
            eh = (struct ether_header *)packet;
            ip = packet_to_ip_hdr(packet);
            hdr = (struct mospf_hdr *)(packet + HDR_OFFSET);
            hello = (struct mospf_hello *)(packet + LSU_OFFSET);
            memcpy(eh->ether_shost, iface->mac, ETH_ALEN);
            memcpy(eh->ether_dhost, "01005E000005", ETH_ALEN);
            eh->ether_type = htons(ETH_P_IP);
            mospf_init_hdr(hdr, MOSPF_TYPE_HELLO, MOSPF_HDR_SIZE + MOSPF_HELLO_SIZE,
    instance->router_id, instance->area_id);
            hdr->checksum = mospf_checksum(hdr);
            mospf_init_hello(hello, iface->mask);
            ip_init_hdr(ip, iface->ip, MOSPF_ALLSPFRouters, length - ETHER_HDR_SIZE,
    IPPROTO_MOSPF);
            iface_send_packet(iface, packet, length);
        }
        pthread_mutex_unlock(&mospf_lock);
    }
    return NULL;
}
```

- 处理 hello 消息

```
void handle_mospf_hello(iface_info_t *iface, const char *packet, int len)
{
    // fprintf(stdout, "TODO: handle mOSPF Hello message.\n");
    struct iphdr *ip = packet_to_ip_hdr(packet);
    struct mospf_hdr *hdr = (struct mospf_hdr *)(packet + HDR_OFFSET);
    struct mospf_hello *hll = (struct mospf_hello *)(packet + LSU_OFFSET);
    mospf_nbr_t *nbr;
    u32 rid = ntohl(hdr->rid);
    list_for_each_entry(nbr, &(iface->nbr_list), list){
        if (nbr->nbr_id == rid){
            goto handle;
        }
    }
    nbr = (mospf_nbr_t *)malloc(sizeof(mospf_nbr_t));
    init_list_head(&nbr->list);
    list_add_tail(&(nbr->list), &(iface->nbr_list));
    iface->num_nbr++;
handle:{
      nbr->nbr_ip = ntohl(ip->saddr);
      nbr->nbr_id = ntohl(hdr->rid);
      nbr->nbr_mask = ntohl(hll->mask);
      nbr->alive = MOSPF_DEFAULT_HELLOINT * 3;
      pthread_mutex_lock(&mospf_lock);
      broadcast_mospf_lsu();
      pthread_mutex_unlock(&mospf_lock);
  }
```

```
}
```

- 定期发送 lsu 消息

```
void *sending_mospf_lsu_thread(void *param)
{
  // fprintf(stdout, "TODO: send mOSPF LSU message periodically.\n");
    while (1){
        sleep(MOSPF_DEFAULT_LSUINT);
        pthread_mutex_lock(&mospf_lock);
        broadcast_mospf_lsu();
        pthread_mutex_unlock(&mospf_lock);
    }
    return NULL;
}
void broadcast_mospf_lsu()
{
    struct mospf_lsa *current;
    mospf_db_entry_t *db;
    fprintf(stdout, "----------\n");
  list_for_each_entry(db, &mospf_db, list){
    fprintf(stdout, ""IP_FMT" %u %u %u\n", HOST_IP_FMT_STR(db->rid), db->seq, db->nadv,
     db->alive);
    for(int i = 0; i < db->nadv; i++){
      current = db->array + i;
      fprintf(stdout, "\t  ");
      fprintf(stdout, IP_FMT, HOST_IP_FMT_STR(current->subnet));
      fprintf(stdout, " ");
      fprintf(stdout, IP_FMT, HOST_IP_FMT_STR(current->mask));
      fprintf(stdout, " ");
      fprintf(stdout, IP_FMT, HOST_IP_FMT_STR(current->rid));
      fprintf(stdout, "\n");
    }
  }
    struct mospf_lsa lsa[8];
    int lsa_size = 8 * MOSPF_LSA_SIZE;
    int number = 0;
    memset(lsa, 0, 8 * MOSPF_LSA_SIZE);
    iface_info_t *iface;
    mospf_nbr_t *nbr;
    list_for_each_entry(iface, &(instance->iface_list), list){
        list_for_each_entry(nbr, &(iface->nbr_list), list){
            lsa[number].subnet = htonl(nbr->nbr_ip);
            lsa[number].mask = htonl(nbr->nbr_mask);
            lsa[number].rid = htonl(nbr->nbr_id);
            number++;
        }
    }
    char *packet;
    struct ether_header *eh;
    struct iphdr *ip;
    struct mospf_hdr *hdr;
    struct mospf_lsu *lsu;
    lsa_size = number * MOSPF_LSA_SIZE;
    int length = LSU_PACKET_SIZE + lsa_size;
    list_for_each_entry(iface, &(instance->iface_list), list){
        list_for_each_entry(nbr, &(iface->nbr_list), list){
            packet = (char *)malloc(length);
            eh = (struct ether_header *)packet;
            ip = packet_to_ip_hdr(packet);
            hdr = (struct mospf_hdr *)(packet + HDR_OFFSET);
            lsu = (struct mospf_lsu *)(packet + LSU_OFFSET);

            eh->ether_type = htons(ETH_P_IP);
            mospf_init_lsu(lsu, number);
            memcpy((char *)(packet + LSU_PACKET_SIZE), lsa, lsa_size);
            mospf_init_hdr(hdr, MOSPF_TYPE_LSU, length - ETHER_HDR_SIZE -
    IP_BASE_HDR_SIZE, instance->router_id, instance->area_id);
```

```
63              hdr->checksum = mospf_checksum(hdr);
64              ip_init_hdr(ip, iface->ip, nbr->nbr_ip, length - ETHER_HDR_SIZE,
    IPPROTO_MOSPF);
65              ip_send_packet(packet, length);
66          }
67      }
68      instance->sequence_num++;
69  }
```

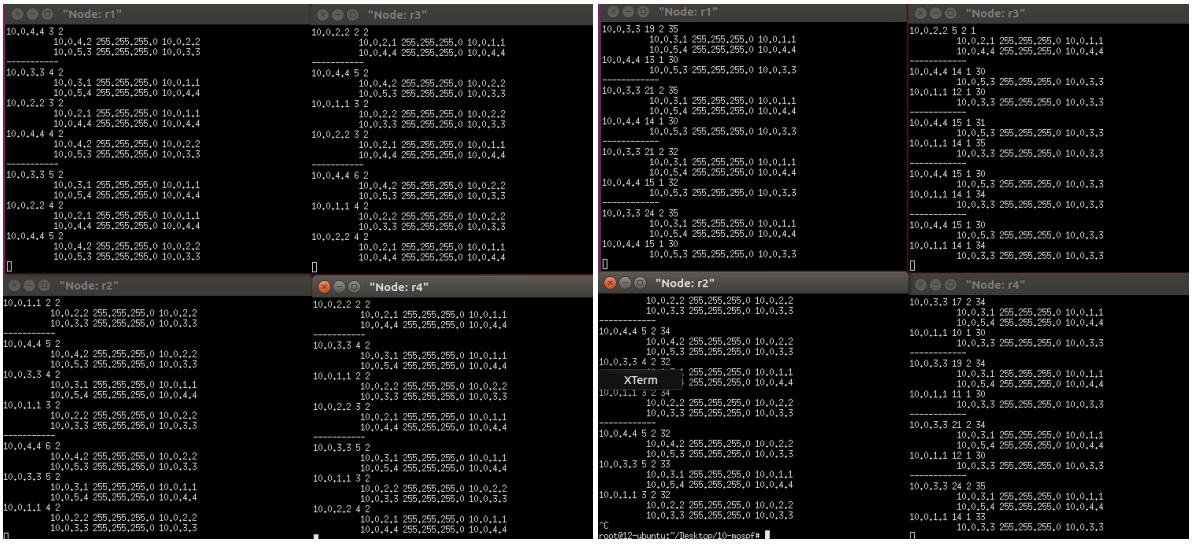
- 处理 lsu 消息

```
1  void handle_mospf_lsu(iface_info_t *iface, char *packet, int len)
2  {
3      // fprintf(stdout, "TODO: handle mOSPF LSU message.\n");
4      struct iphdr *ip = packet_to_ip_hdr(packet);
5      struct mospf_hdr *hdr = (struct mospf_hdr *)(packet + HDR_OFFSET);
6      struct mospf_lsu *lsu = (struct mospf_lsu *)(packet + LSU_OFFSET);
7      int nadv = ntohl(lsu->nadv);
8      u32 rid = ntohl(hdr->rid);
9      u16 seq = ntohs(lsu->seq);
10     mospf_db_entry_t *db;
11     if(rid == instance->router_id){
12       return;
13     }
14     list_for_each_entry(db, &mospf_db, list){
15         if (db->rid == rid){
16             if(seq > db->seq){
17                 free(db->array);
18                 goto handle;
19             }else{
20                 return;
21             }
22         }
23     }
24     db = (mospf_db_entry_t *)malloc(sizeof(mospf_db_entry_t));
25     db->rid = ntohl(hdr->rid);
26     init_list_head(&db->list);
27     list_add_tail(&(db->list), &mospf_db);
28 handle:{
29     db->alive = MOSPF_DATABASE_TIMEOUT;
30     db->seq = ntohs(lsu->seq);
31     db->nadv = nadv;
32     db->array = (struct mospf_lsa *)malloc(nadv * MOSPF_LSA_SIZE);
33       struct mospf_lsa *current, *aim;
34       for (int i = 0; i < nadv; i++){
35           current = db->array + i;
36           aim = (struct mospf_lsa *)((char *)lsu + MOSPF_LSU_SIZE) + i;
37           current->subnet = ntohl(aim->subnet);
38           current->mask = ntohl(aim->mask);
39           current->rid = ntohl(aim->rid);
40       }
41       char *pkt_new;
42       mospf_nbr_t *nbr;
43       iface_info_t *ifaces;
44       if(--lsu->ttl > 0){
45           list_for_each_entry(ifaces, &(instance->iface_list), list)
46           {
47               if(ifaces->index != iface->index){
48                 list_for_each_entry(nbr, &(ifaces->nbr_list), list)
49                 {
50                     pkt_new = (char *)malloc(len);
51                     ip = packet_to_ip_hdr(pkt_new);
52                     hdr = (struct mospf_hdr *)(pkt_new + HDR_OFFSET);
53                     memcpy(pkt_new, packet, len);
54
55                     hdr->checksum = mospf_checksum(hdr);
56                     ip_init_hdr(ip, ifaces->ip, nbr->nbr_ip, len - IP_BASE_HDR_SIZE +
    MOSPF_HDR_SIZE + MOSPF_LSA_SIZE + nadv * MOSPF_LSU_SIZE, IPPROTO_MOSPF);
```

```
57                          ip_send_packet(pkt_new, len);
58                      }
59                  }
60                      }
61              }
62      }
63  }
```

## 4 实验结果

- 输出为 rid、seq、nadv、alive 及 array 中每个 mospf_lsa 的 subnet、mask、rid
- 左图为关掉协议后，运行 mospfd，各节点生成一致的链路状态数据库
- 右图为将 r2 的 mospfd 终止后，拓扑发生变动及数据库老化后，生成新的一致的链路状态数据库




## 5 结果分析

- 左图，各节点通过发送 hello 包及 lsu 消息，较快的生成一致的链路状态数据库
- 右图，r1、r4 较快的更新掉无用数据，
- 而 r3 等待数据库老化时间后与 r1、r4 生成一致的链路状态数据库
- r1、r4 与 r2 直接连接，能直接通过传递消息，在 r2 关闭后及时生成新的链路状态数据库
- r3 没有与 r2 直接相连，其与 r2 的链路状态数据库，由数据库老化操作，一定时间后更新