计算机网络研讨课实验报告

冯吕 2015K8009929049

2018 年 5 月 25 日

实验题目

网络路由实验一。

实验内容

本次实验需要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。

实验流程

在实验中，需要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，需要实现如下 5 个函数:

0.1 *sending_mospf_hello_thread* 函数

该函数实现节点广播自己，周期性广播 (5s): 发送 *mOSPF Hello* 消息，包括节点 *ID*, 端口的子网掩码目的 IP 地址为 224.0.0.5, 目的 MAC 地址为 $01:00:5E:00:00:05$。

```
1  void *sending_mospf_hello_thread(void *param)
2  {
3          fprintf(stdout, "TODO: send mOSPF Hello message periodically.\n");
4          while (1){
5                  sleep(MOSPF_DEFAULT_HELLOINT);
6                  pthread_mutex_lock(&mospf_lock);
7                  iface_info_t *iface;
8                  list_for_each_entry(iface, &instance->iface_list, list){
9                          int len = ETHER_HDR_SIZE + IP_BASE_HDR_SIZE +
10                         MOSPF_HDR_SIZE + MOSPF_HELLO_SIZE;
11                         char *hello_packet = (char *)malloc(len);
12                         struct ether_header *eth = (struct ether_header *)
13                         hello_packet;
14                         memcpy(eth->ether_shost, iface->mac, ETH_ALEN);
15                         u8 dhost[ETH_ALEN] = {0x01, 0x00, 0x5e, 0x00, 0x00,
16                         0x05};
17                         memcpy(eth->ether_dhost, dhost, ETH_ALEN);
```

```
18                          eth->ether_type = htons(ETH_P_IP);
19
20                          struct iphdr *iph = packet_to_ip_hdr(hello_packet);
21                          ip_init_hdr(iph, iface->ip, 0xe0000005, len -
22                          ETHER_HDR_SIZE, 90);
23
24                          struct mospf_hdr *mohdr = (struct mospf_hdr *)
25                          (hello_packet + ETHER_HDR_SIZE + IP_BASE_HDR_SIZE);
26                          mospf_init_hdr(mohdr, 0x1, MOSPF_HDR_SIZE +
27                          MOSPF_HELLO_SIZE, instance->router_id, 0);
28
29                          struct mospf_hello *hello = (struct mospf_hello *)
30  (hello_packet + ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + MOSPF_HDR_SIZE);
31                          mospf_init_hello(hello, iface->mask);
32                          mohdr->checksum = mospf_checksum(mohdr);
33                          iface_send_packet(iface, hello_packet, len);
34                  }
35                  pthread_mutex_unlock(&mospf_lock);
36          }
37          return NULL;
38  }
```

0.2 *checking_nbr_thread* 函数

该函数实现邻居列表的老化操作，如果列表中的节点在 $3 * hello - interval$ 时间内未更新，则将其删除。

```
1   void *checking_nbr_thread(void *param)
2   {
3           fprintf(stdout, "TODO: neighbor list timeout operation.\n");
4           while (1){
5                   sleep(1);
6                   pthread_mutex_lock(&mospf_lock);
7                   iface_info_t *iface;
8                   list_for_each_entry(iface, &instance->iface_list, list){
9                           mospf_nbr_t *mos_pos, *mos_q;
10                          list_for_each_entry_safe(mos_pos, mos_q,
11                          &iface->nbr_list, list){
12                                  if (mos_pos->alive > 3 *
13                                  MOSPF_DEFAULT_HELLOINT){
14                                          list_delete_entry(&mos_pos->list);
15                                          free(mos_pos);
16                                          --iface->num_nbr;
17                                  }
18                                  else {
```

```
                                        ++mos_pos->alive;
                                }
                        }
                }
                pthread_mutex_unlock(&mospf_lock);
        }
        return NULL;
}
```

0.3 *handle_mospf_hello* 函数

该函数处理 *mOSPF Hello* 消息:

- 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)
{
        fprintf(stdout, "TODO: handle mOSPF Hello message.\n");
        pthread_mutex_lock(&mospf_lock);
        struct iphdr *iph = packet_to_ip_hdr(packet);
        struct mospf_hdr *moph = (struct mospf_hdr *)(packet +
        IP_BASE_HDR_SIZE + ETHER_HDR_SIZE);
        struct mospf_hello *hello = (struct mospf_hello *)(packet +
        ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + MOSPF_HDR_SIZE);

        u32 hello_ip = ntohl(iph->saddr);
        u32 hello_rid = ntohl(moph->rid);
        u32 hello_mask = ntohl(hello->mask);
        mospf_nbr_t *mos_nbr;
        int flag = 0;
        list_for_each_entry(mos_nbr, &iface->nbr_list, list){
                if (mos_nbr->nbr_ip == hello_ip){
                        flag = 1;
                        mos_nbr->alive = 0;
                }
        }
        if (!flag){
                mospf_nbr_t *new_nbr = (mospf_nbr_t *)
                malloc(sizeof(mospf_nbr_t));
                new_nbr->nbr_id = hello_rid;
                new_nbr->nbr_ip = hello_ip;
                new_nbr->nbr_mask = hello_mask;
                new_nbr->alive = 0;
                list_add_tail(&new_nbr->list, &iface->nbr_list);
```

```
30                  ++iface->num_nbr;
31          }
32          pthread_mutex_unlock(&mospf_lock);
33  }
```

0.4 *sending_mospf_lsu_thread* 函数

该函数实现节点向每个邻居节点发送链路状态信息，信息包含的内容有：

- 该节点 *ID*(*mOSPF Header*)、邻居节点 *ID*、网络和掩码 (*mOSPF LSU*)
- 序列号 (*sequence number*), 每次生成链路状态信息时加 1;
- 目的 IP 地址为邻居节点相应端口的 *IP* 地址, 目的 *MAC* 地址为邻居节点相应端口的 *MAC* 地址;

```
1   void *sending_mospf_lsu_thread(void *param)
2   {
3           fprintf(stdout, "TODO: send mOSPF LSU message periodically.\n");
4       while(1){
5           sleep(MOSPF_DEFAULT_LSUINT);
6
7           pthread_mutex_lock(&mospf_lock);
8           int num_adv = 0;
9           iface_info_t *iface;
10          list_for_each_entry(iface, &(instance->iface_list), list){
11                  if(iface->num_nbr == 0)
12                          num_adv ++;
13                  else
14                  num_adv += iface->num_nbr;
15          }
16          int pac_len = ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + MOSPF_HDR_SIZE
17                  + MOSPF_LSU_SIZE + num_adv * MOSPF_LSA_SIZE;
18          char *packet = (char *)malloc(pac_len);
19          mospf_nbr_t *mos_nbr;
20          struct mospf_lsa *mos_lsa;
21          int i = 0;
22          list_for_each_entry(iface, &(instance->iface_list), list){
23                  if(iface->num_nbr == 0){
24                          mos_lsa = (struct mospf_lsa *)(packet + pac_len
25                                  - (num_adv - i) * MOSPF_LSA_SIZE);
26                          ++i;
27                          mos_lsa->subnet = htonl(iface->ip & iface->mask);
28                          mos_lsa->mask = htonl(iface->mask);
29                          mos_lsa->rid = 0;
30                          continue;
31                  }
```

```
32              list_for_each_entry(mos_nbr, &(iface->nbr_list), list){
33                  mos_lsa = (struct mospf_lsa *)(packet + pac_len
34                                  - (num_adv - i) * MOSPF_LSA_SIZE);
35                  ++i;
36                  mos_lsa->subnet = ntohl(mos_nbr->nbr_ip & mos_nbr->nbr_mask);
37                  mos_lsa->mask = ntohl(mos_nbr->nbr_mask);
38                  mos_lsa->rid = ntohl(mos_nbr->nbr_id);
39              }
40          }
41          list_for_each_entry(iface, &(instance->iface_list), list){
42              list_for_each_entry(mos_nbr, &(iface->nbr_list), list){
43                  char *packet_t = (char *)malloc(pac_len);
44                  memcpy(packet_t, packet, pac_len);
45                  struct ether_header *eth = (struct ether_header *)packet_t;
46                  eth->ether_type = htons(ETH_P_IP);
47                  memcpy(eth->ether_shost, iface->mac, ETH_ALEN);
48
49                  struct iphdr *iph = (struct iphdr *)(packet_t
50                                  + ETHER_HDR_SIZE);
51                  ip_init_hdr(iph, iface->ip, mos_nbr->nbr_ip, pac_len
52                                  - ETHER_HDR_SIZE, 90);
53
54                  struct mospf_hdr * mospf = (struct mospf_hdr *)(packet_t
55                                  + IP_BASE_HDR_SIZE + ETHER_HDR_SIZE);
56                  mospf_init_hdr(mospf, MOSPF_TYPE_LSU, pac_len
57                                  - ETHER_HDR_SIZE - IP_BASE_HDR_SIZE, instance->router_
58                          instance->area_id);
59
60
61                  struct mospf_lsu *mos_lsu = (struct mospf_lsu *)
62                                  ((char *)mospf + MOSPF_HDR_SIZE);
63                  mospf_init_lsu(mos_lsu, num_adv);
64                  mospf->checksum = mospf_checksum(mospf);
65                  ip_send_packet(packet_t, pac_len);
66              }
67
68          }
69          instance->sequence_num ++;
70          free(packet);
71          pthread_mutex_unlock(&mospf_lock);
72      }
73          return NULL;
74  }
```

0.5 *handle_mospf_lsu* 函数

该函数处理 *mOSF LSU* 消息:

- 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;
- *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

```
1  void handle_mospf_lsu(iface_info_t *iface, char *packet, int len)
2  {
3          fprintf(stdout, "TODO: handle mOSPF LSU message.\n");
4      mospf_db_entry_t *mos_db_en;
5      int flag = 0;
6      struct mospf_hdr * mospf = (struct mospf_hdr *)(packet
7          + IP_BASE_HDR_SIZE + ETHER_HDR_SIZE);
8      struct mospf_lsu *mos_lsu = (struct mospf_lsu *)((char *)mospf
9          + MOSPF_HDR_SIZE);
10     struct mospf_lsa *mos_lsa = (struct mospf_lsa *)((char *)mos_lsu
11         + MOSPF_LSU_SIZE);
12     int mospf_rid = ntohl(mospf->rid);
13     fprintf(stdout, IP_FMT"\t\n",
14                                         HOST_IP_FMT_STR(mospf_rid)
15                                     );
16     if(instance->router_id == mospf_rid) return;
17     int seq_num = ntohs(mos_lsu->seq);
18     int nadv = ntohl(mos_lsu->nadv);
19     list_for_each_entry(mos_db_en, &(mospf_db), list){
20         if(mospf_rid == mos_db_en->rid && mos_db_en->seq >= seq_num)
21                 flag = 1;
22         else if(mospf_rid == mos_db_en->rid && mos_db_en->seq < seq_num){
23                 flag = 1;
24                 free(mos_db_en->array);
25                 mos_db_en->array = (struct mospf_lsa *)malloc(nadv
26                         * MOSPF_LSA_SIZE);
27                 for(int i = 0; i < nadv; i++){
28                     struct mospf_lsa *lsa = (struct mospf_lsa *)(
29                                 (char *)mos_lsa + i * MOSPF_LSA_SIZE);
30                     mos_db_en->array[i].rid = ntohl(lsa->rid);
31                     mos_db_en->array[i].subnet = ntohl(lsa->subnet);
32                     mos_db_en->array[i].mask = ntohl(lsa->mask);
33             }
34         }
35     }
36     if(! flag){
37         mospf_db_entry_t * mospf_db_en_t = (mospf_db_entry_t *)
38                 malloc(sizeof(mospf_db_entry_t));
39         mospf_db_en_t->rid = mospf_rid;
```

```
40          mospf_db_en_t->seq = seq_num;
41          mospf_db_en_t->nadv = nadv;
42
43          mospf_db_en_t->array = (struct mospf_lsa *)malloc(nadv
44                  * MOSPF_LSA_SIZE);
45          for(int i = 0; i < nadv; i++){
46                  struct mospf_lsa *lsa = (struct mospf_lsa *)((char
47                          *)mos_lsa + i * MOSPF_LSA_SIZE);
48                  mospf_db_en_t->array[i].rid = ntohl(lsa->rid);
49                  mospf_db_en_t->array[i].subnet = ntohl(lsa->subnet);
50                  mospf_db_en_t->array[i].mask = ntohl(lsa->mask);
51          }
52          list_add_tail(&(mospf_db_en_t->list), &mospf_db);
53      }
54      mospf_db_entry_t *mosdb;
55      list_for_each_entry(mosdb, &mospf_db, list){
56          for(int i = 0;i < mosdb->nadv; i++)
57                  fprintf(stdout, IP_FMT"\t"IP_FMT"\t"IP_FMT"\t"IP_FMT"\n",
58                          HOST_IP_FMT_STR(mosdb->rid),
59                                  HOST_IP_FMT_STR(mosdb->array[i].subnet),
60                                  HOST_IP_FMT_STR(mosdb->array[i].mask),
61                                  HOST_IP_FMT_STR(mosdb->array[i].rid)
62                                  );
63      }
64
65      mos_lsu->ttl -= 1;
66      if(mos_lsu->ttl > 0){
67          iface_info_t *iface_t;
68          mospf_nbr_t *mos_nbr;
69          list_for_each_entry(iface_t, &(instance->iface_list), list){
70                  if(iface_t->index == iface->index)
71                          continue;
72              list_for_each_entry(mos_nbr, &(iface_t->nbr_list), list){
73                  char *packet_t = (char *)malloc(len);
74                  memcpy(packet_t, packet, len);
75                  struct ether_header *eth = (struct ether_header *)packet_t;
76                  memcpy(eth->ether_shost, iface_t->mac, ETH_ALEN);
77                  struct iphdr *iph = (struct iphdr *)(packet_t
78                                  + ETHER_HDR_SIZE);
79                  struct mospf_hdr * mospf = (struct mospf_hdr *)(packet_t
80                                  + IP_BASE_HDR_SIZE + ETHER_HDR_SIZE);
81                  mospf->checksum = mospf_checksum(mospf);
82                  ip_init_hdr(iph, iface_t->ip, mos_nbr->nbr_ip, len
83                                  - ETHER_HDR_SIZE, 90);
```

```
84                      ip_send_packet(packet_t, len);
85                  }
86              }
87          }
88      }
```

0.6 运行实验

之后，运行实验，在各个路由器节点上运行 $mospfd$，使各个节点生成一致的链路状态数据库。

实验结果

各个路由器节点生成了一致的链路状态数据库：

图 1: 运行截图

结果分析

通过洪泛机制，各个路由器能够生成一致的链路状态数据库，实验结果正确。