

计算机网络实验报告-11

阮星程

2015K8009929047

一、 实验题目

网络路由实验一。

二、 实验内容

理解 OSPF 协议构建路由表的原理，完成构建的第一步——构建一致性链路状态数据库。

三、 实验过程

本次实验的主要代码在程序 `mospf_daemon.c` 中，分为五个部分。发送 hello 消息，处理 hello 包，处理超时的邻居节点信息，发送 lsu 消息，处理接受到的 lsu 消息，但其实还应该加入对应的 lsu 超时处理才对，鉴于头文件中的 `db_entry` 结构体没有相应处理超时的条目，所以就暂未处理。

接下来逐步构建各个函数。

发送 hello 消息。

对于需要间隔发送的条目，我们自然地使用 `while(1)`和 `sleep` 的组合，之后开始组合 packet。

```
eth->ether_dhost[0] = 0x01;
eth->ether_dhost[1] = 0x00;
eth->ether_dhost[2] = 0x5E;
eth->ether_dhost[3] = 0x00;
eth->ether_dhost[4] = 0x00;
eth->ether_dhost[5] = 0x05;
eth->ether_type = htons(ETH_P_IP);

ip->version = 4;
ip->ihl = 5;
ip->tos = 0;
ip->tot_len = htons(IP_BASE_HDR_SIZE + MOSPF_HDR_SIZE + MOSPF_HELLO_SIZE);
ip->id = htons(0);
ip->frag_off = 0;
ip->ttl = DEFAULT_TTL;
ip->protocol = 90;
ip->daddr = htonl(0xE0000005);

mospf->version = MOSPF_VERSION;
mospf->type = MOSPF_TYPE_HELLO;
mospf->len = htons(MOSPF_HDR_SIZE + MOSPF_HELLO_SIZE);
mospf->rid = htonl(instance->router_id);
mospf->aid = htonl(instance->area_id);
mospf->padding = htons(0);

hello->helloint = htons(MOSPF_DEFAULT_HELLOINT);
hello->padding = htons(0);
```

先初始化一部分共有的数据，然后再使用循环，将私有的部分填充起来

```

//...
iface_info_t * iface;
list_for_each_entry(iface, &instance->iface_list, list){
    char * iface_packet = (char *) malloc(ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + MOSPF_HDR_SIZE);
    memcpy(iface_packet, packet, ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + MOSPF_HDR_SIZE + MOSPF_H
    struct ether_header * eth = (struct ether_header *)iface_packet;
    struct iphdr *ip = (struct iphdr *)((char *)iface_packet + ETHER_HDR_SIZE);
    struct mospf_hdr *mospf = (struct mospf_hdr *)((char *)ip + IP_BASE_HDR_SIZE);
    struct mospf_hello *hello = (struct mospf_hello *)((char *)mospf + MOSPF_HDR_SIZE);
    hello->mask = htonl(iface->mask);
    mospf->checksum = mospf_checksum(mospf);
    ip->saddr = htonl(iface->ip);
    ip->checksum = ip_checksum(ip);
    memcpy(eth->ether_shost, iface->mac, 6);
    iface_send_packet(iface, iface_packet, ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + MOSPF_HDR_SIZE
}
free(packet);

```

这样，发送 hello 消息的部分就算是构建完成了，没有使用各个 init 函数是因为各共有私有数据不大好直接利用 init 函数书写出来，全部放在私有函数里 init 又觉得有些浪费计算能力，所以做了这样的优化。

接下来是处理 timeout 的函数，本函数就相应简单，只需要每秒将每个 neighbor 条目的 alive time 减 1，为 0 时删除即可。

```

void *checking_nbr_thread(void *param)
{
    //fprintf(stdout, "TODO: neighbor list timeout operation.\n");
    mospf_nbr_t * nbr, * nbr1;
    iface_info_t * iface;
    while(1){
        sleep(1);
        pthread_mutex_lock(&mospf_lock);
        list_for_each_entry(iface, &instance->iface_list, list){
            list_for_each_entry_safe(nbr, nbr1, &iface->nbr_list, list){
                if(--nbr->alive == 0){
                    list_delete_entry((struct list_head *)nbr);
                    iface->num_nbr--;
                    nbr_changed = 1;
                }
            }
        }
        pthread_mutex_unlock(&mospf_lock);
    }

    return NULL;
}

```

处理 hello 程序的部分较为简单，将包中的信息添加到对应的 iface 中即可，也就不再叙述了。

发送 lsu 包的程序就稍显复杂了，由于我们每个端口中都存有各自的邻居信息，所以我们需要先将这些信息整合起来然后再进行组包，整合过程如下：

```

//gather nbr informations, save it to database
struct mospf_lsa * lsa_list;
struct mospf_lsa * lsa_tmp;
int nadv = 0;
list_for_each_entry(iface, &instance->iface_list, list){
    nadv += iface->num_nbr;
}

lsa_list = (struct mospf_lsa *)malloc(MOSPF_LSA_SIZE * nadv);
lsa_tmp = lsa_list;
list_for_each_entry(iface, &instance->iface_list, list){
    list_for_each_entry(nbr, &iface->nbr_list, list){
        lsa_tmp->subnet = nbr->nbr_ip;
        lsa_tmp->mask = nbr->nbr_mask;
        lsa_tmp->rid = nbr->nbr_id;
        lsa_tmp++;
    }
}

int found = 0;
list_for_each_entry(db, &mospf_db, list){
    if(db->rid == instance->router_id){
        found = 1;
        db->array = lsa_list;
        db->seq = instance->sequence_num;
        db->nadv = nadv;
        break;
    }
}

if(!found){
    db = (mospf_db_entry_t *)malloc(sizeof(mospf_db_entry_t));
    db->array = lsa_list;
    init_list_head(&db->list);
    db->nadv = nadv;
    db->rid = instance->router_id;
    db->seq = instance->sequence_num;
    list_add_tail((struct list_head *) db, &mospf_db);
}

```

首先遍历一遍各表，然后得到邻居的数目，在 malloc 相应的空间，然后添加各个条目。之后我们就能开始组包发包了，后面的流程和 hello 包重复较多，也就不再展示了。比较有趣的部分是我们怎么实现邻居节点变动时发送 lsu 消息，另写一个线程的话较为浪费，比较合适的做法还是像下图中这样，增加一个 changed 的全局变量，然后每一秒检测一次是否这个变量为 1 了，如果为 1 那么就跳出 sleep 的循环然后开始下面的流程。

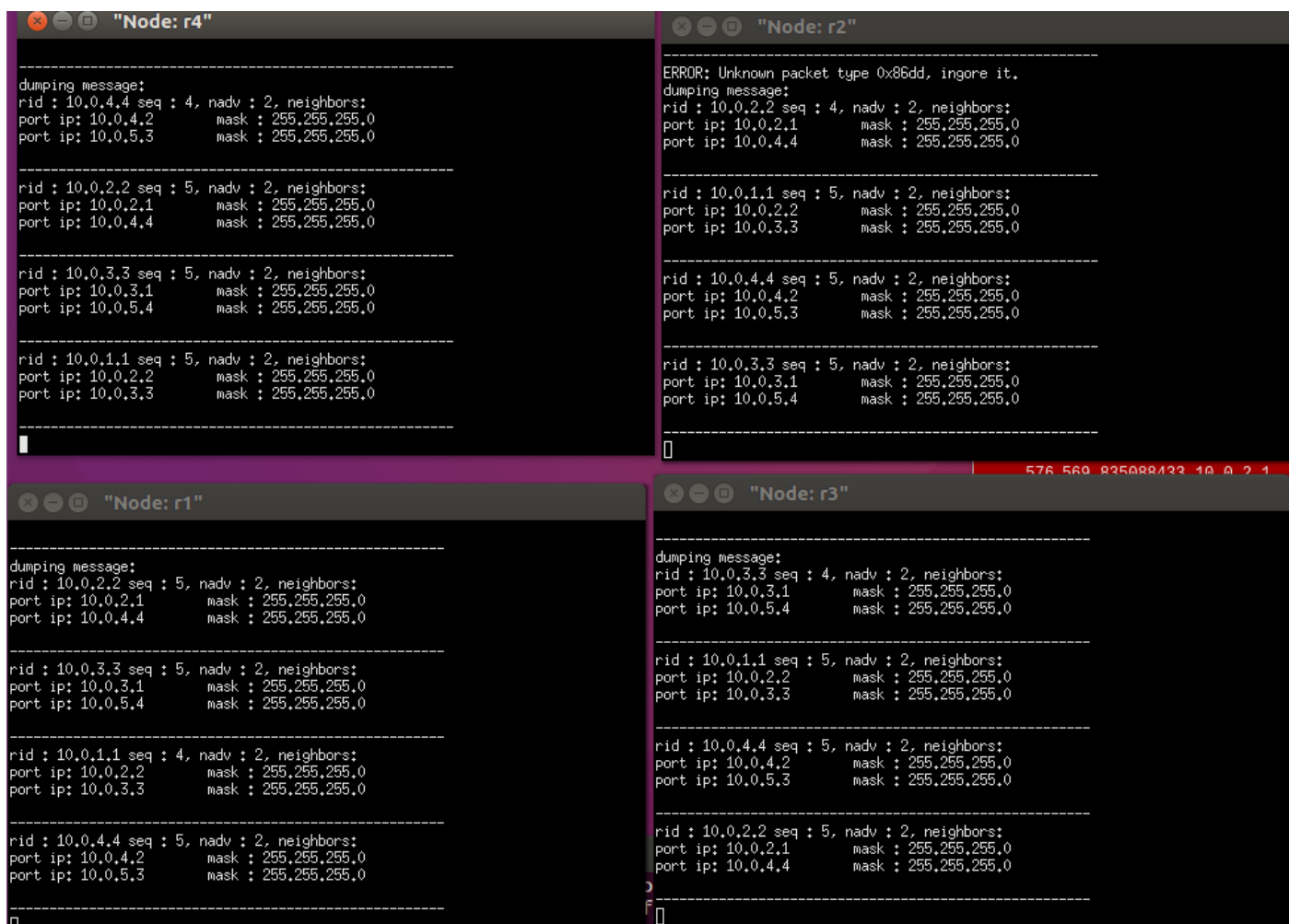
```

pthread_mutex_lock(&mospf_lock);
while(!nbr_changed && --left_interval){
    pthread_mutex_unlock(&mospf_lock);
    sleep(1);
    pthread_mutex_lock(&mospf_lock);
}
nbr_changed = 0;
left_interval = MOSPF_DEFAULT_LSUINT;

```

Handle 的部分同样，仅仅比 hello 包的处理多了一个 forward 的过程，不再赘述，只提一点在实验抓包时发现如果只使用 ttl，那么环路转发的现象会显得较为严重，浪费了很多的计算能力以及链路带宽，所以对于 seq 不比当前存储大的包选择直接丢弃即可。

四、 实验结果



```
"Node: r4"
-----
dumping message:
rid : 10.0.4.4 seq : 4, nadv : 2, neighbors:
port ip: 10.0.4.2      mask : 255.255.255.0
port ip: 10.0.5.3      mask : 255.255.255.0

-----
rid : 10.0.2.2 seq : 5, nadv : 2, neighbors:
port ip: 10.0.2.1      mask : 255.255.255.0
port ip: 10.0.4.4      mask : 255.255.255.0

-----
rid : 10.0.3.3 seq : 5, nadv : 2, neighbors:
port ip: 10.0.3.1      mask : 255.255.255.0
port ip: 10.0.5.4      mask : 255.255.255.0

-----
rid : 10.0.1.1 seq : 5, nadv : 2, neighbors:
port ip: 10.0.2.2      mask : 255.255.255.0
port ip: 10.0.3.3      mask : 255.255.255.0

-----

"Node: r2"
-----
ERROR: Unknown packet type 0x86dd, ignore it.
dumping message:
rid : 10.0.2.2 seq : 4, nadv : 2, neighbors:
port ip: 10.0.2.1      mask : 255.255.255.0
port ip: 10.0.4.4      mask : 255.255.255.0

-----
rid : 10.0.1.1 seq : 5, nadv : 2, neighbors:
port ip: 10.0.2.2      mask : 255.255.255.0
port ip: 10.0.3.3      mask : 255.255.255.0

-----
rid : 10.0.4.4 seq : 5, nadv : 2, neighbors:
port ip: 10.0.4.2      mask : 255.255.255.0
port ip: 10.0.5.3      mask : 255.255.255.0

-----
rid : 10.0.3.3 seq : 5, nadv : 2, neighbors:
port ip: 10.0.3.1      mask : 255.255.255.0
port ip: 10.0.5.4      mask : 255.255.255.0

-----

"Node: r1"
-----
dumping message:
rid : 10.0.2.2 seq : 5, nadv : 2, neighbors:
port ip: 10.0.2.1      mask : 255.255.255.0
port ip: 10.0.4.4      mask : 255.255.255.0

-----
rid : 10.0.3.3 seq : 5, nadv : 2, neighbors:
port ip: 10.0.3.1      mask : 255.255.255.0
port ip: 10.0.5.4      mask : 255.255.255.0

-----
rid : 10.0.1.1 seq : 4, nadv : 2, neighbors:
port ip: 10.0.2.2      mask : 255.255.255.0
port ip: 10.0.3.3      mask : 255.255.255.0

-----
rid : 10.0.4.4 seq : 5, nadv : 2, neighbors:
port ip: 10.0.4.2      mask : 255.255.255.0
port ip: 10.0.5.3      mask : 255.255.255.0

-----

"Node: r3"
-----
dumping message:
rid : 10.0.3.3 seq : 4, nadv : 2, neighbors:
port ip: 10.0.3.1      mask : 255.255.255.0
port ip: 10.0.5.4      mask : 255.255.255.0

-----
rid : 10.0.1.1 seq : 5, nadv : 2, neighbors:
port ip: 10.0.2.2      mask : 255.255.255.0
port ip: 10.0.3.3      mask : 255.255.255.0

-----
rid : 10.0.4.4 seq : 5, nadv : 2, neighbors:
port ip: 10.0.4.2      mask : 255.255.255.0
port ip: 10.0.5.3      mask : 255.255.255.0

-----
rid : 10.0.2.2 seq : 5, nadv : 2, neighbors:
port ip: 10.0.2.1      mask : 255.255.255.0
port ip: 10.0.4.4      mask : 255.255.255.0

-----
```

可见上图，各个路由器都得到了对应的链路数据库，结果正确。

五、 实验总结

本次实验其实从逻辑上来讲比较容易，比较麻烦的就是构建包的整个过程比较繁杂，原因之一肯定是有没有使用 init 函数，但是为了计算效率，稍微牺牲一些程序的长度也无可厚非。另外困扰了我一段时间的问题是 iface 中存储的内容与 db 中项的关系，在刚刚开始的时候我没有想清楚，直到实验构造到发 lsu 包的时候这个问题才体现出来。稍微花了几分钟修改了下之前的构造，才算是成功解决问题吧。调试的话没有遇上什么问题，很快便解决了，有了 wireshark 之后调试效率确实也高了不少，另外对网络序和字节序的转换也变得相对熟悉了，在这次实验中没有出现什么问题，也算是令人欣慰的一点吧。