10-OSPF实验–report

作者：苗屹松

日期：11月26日2017年

1. Overview

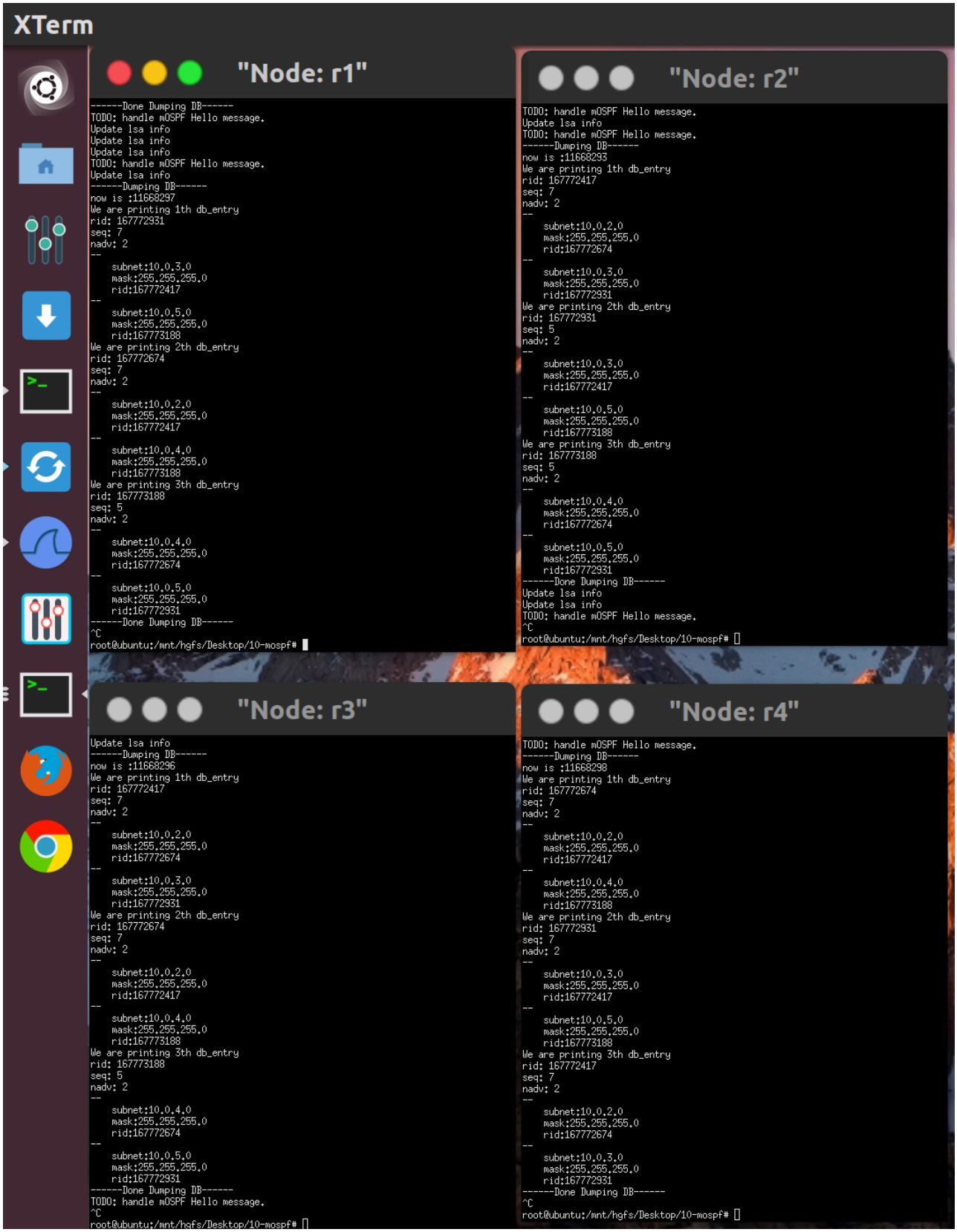
这次实验完成了OSPF协议，借助老师的框架代码，我了解了OSPF包的格式，同时也知道在那些时刻产生并发送相应的OSPF包(hello, LSU)

这个实验的另一个收获是我对router有了更好的了解，将在下文描述。

2. 实验结果展示

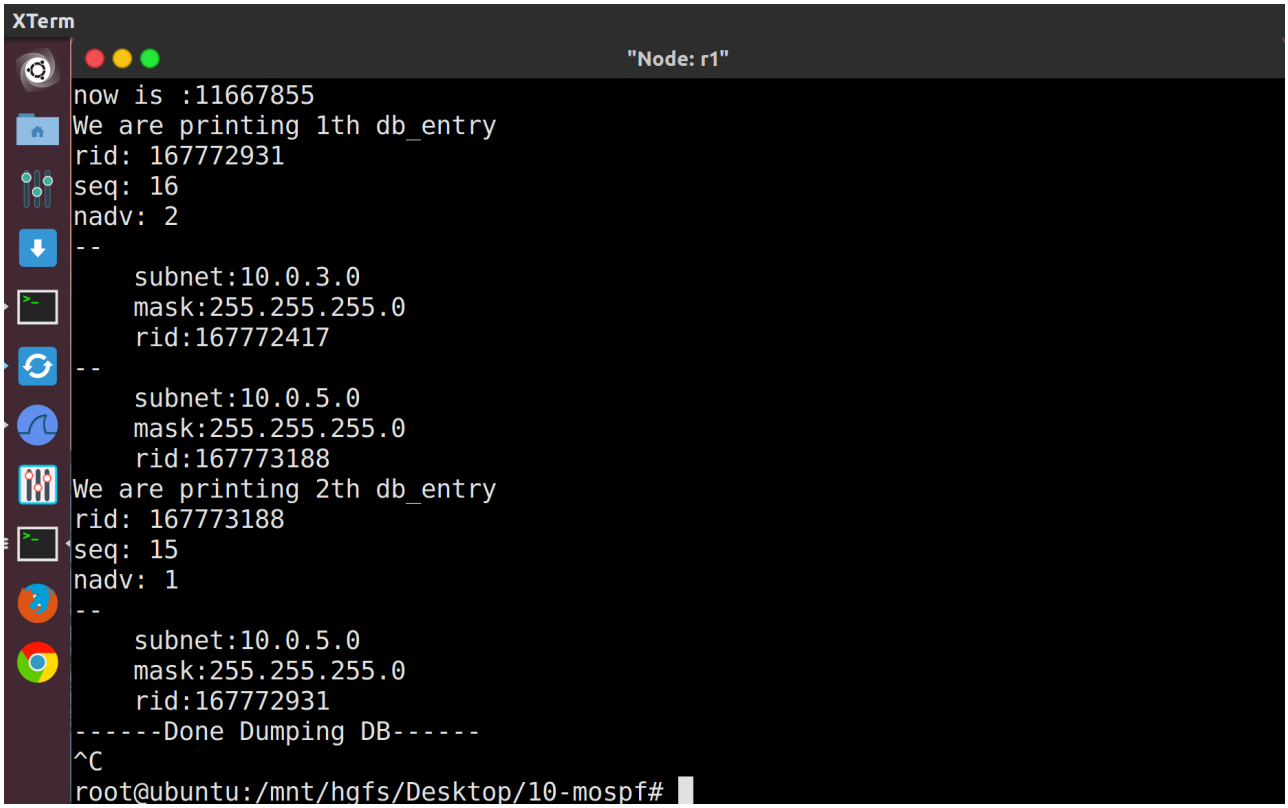
按照实验的要求，我通过了所有的测试！！

2.1 r1, r2, r3, r4都运行mospfd

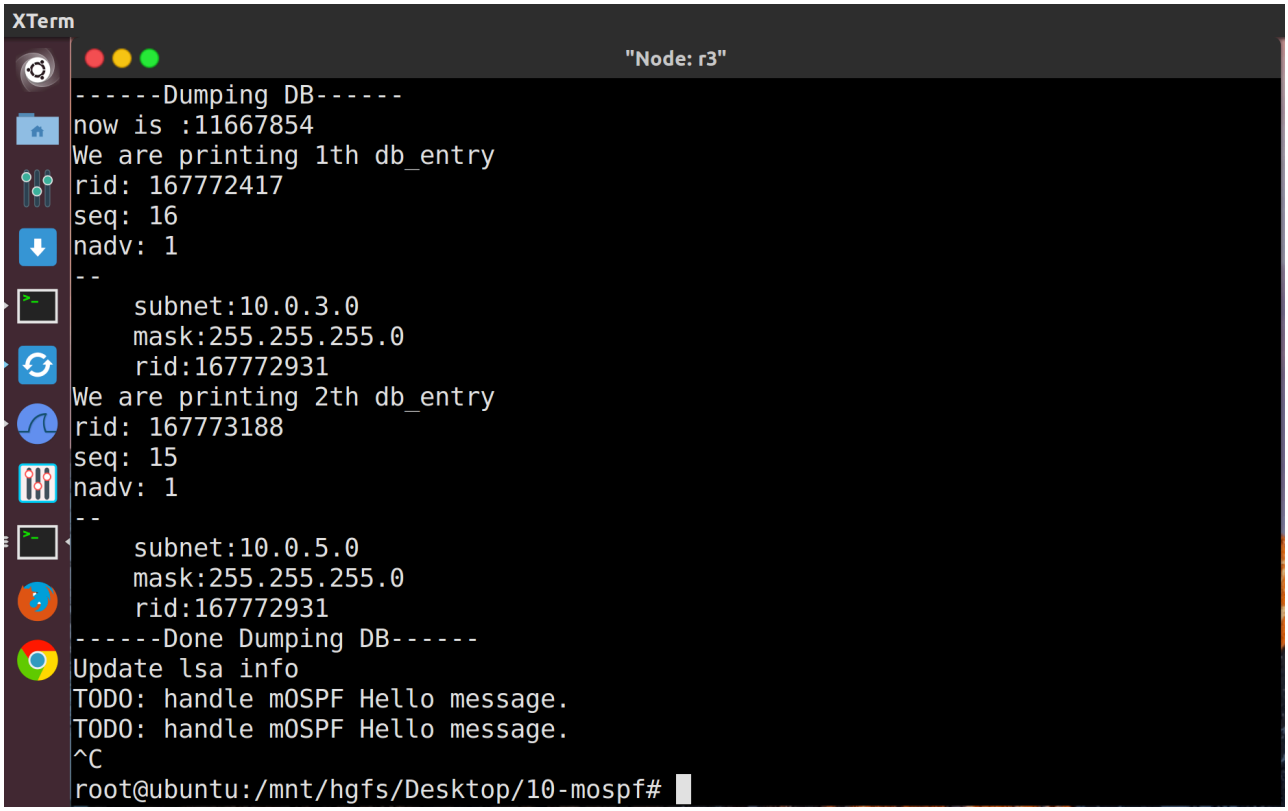


我们可以看到r1~r4的都有除了自己的所有链路信息！ 比如说，r1中有r2, r3, r4的链路信息。而且这些信息在r1~r4中都是一致的！所以这就形成了一致的链路状态数据库。

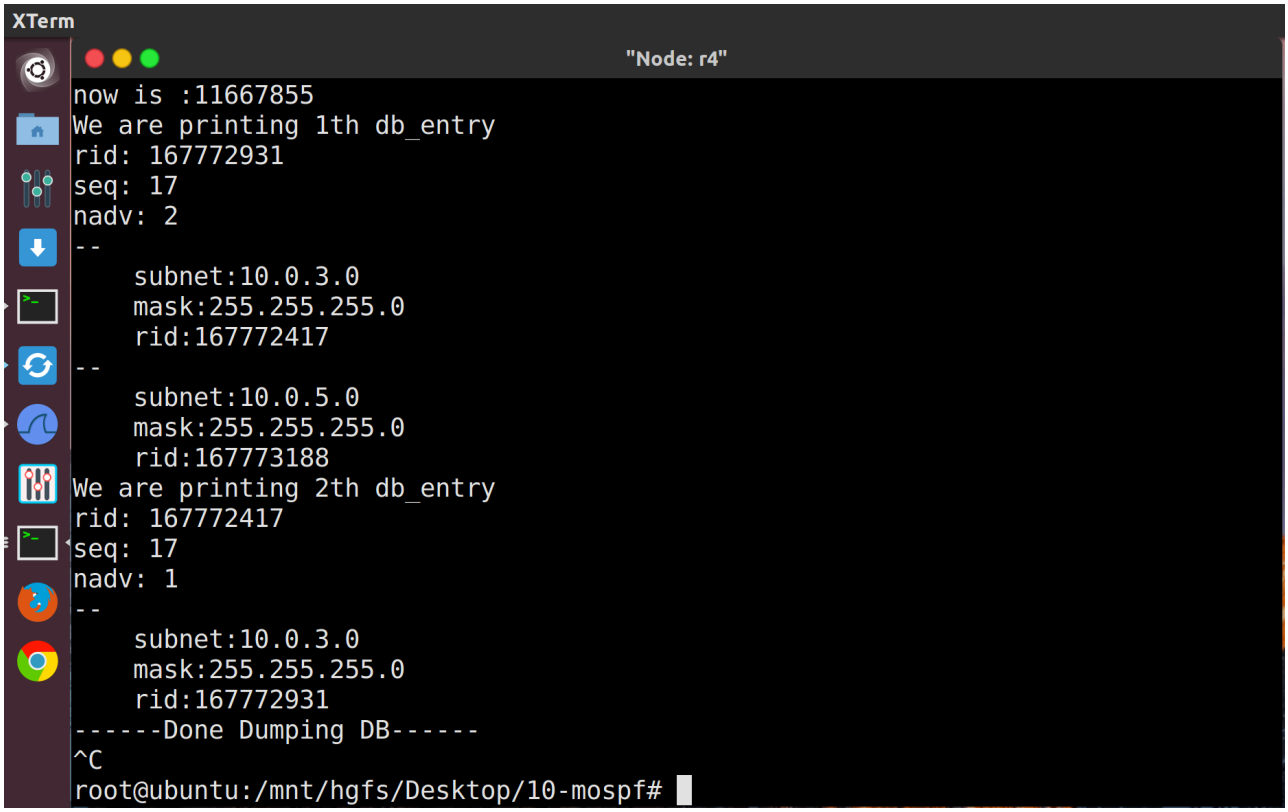
2.2 先在r1~r4上运行 mospfd， 然后关掉r2



(r1的链路状态数据库)



(r3的链路状态数据库)



(r4的链路状态数据库)

我们稍加观察就可知道这些链路状态是完全正确的！其中r2的信息已被删掉(通过新加入的老化操作)

3. 文件描述 与 操作指南

`10-ospf-yisong-report.pdf`: 实验报告

`\log\r*_log.txt`: 运行时的输出，每5秒dump一次链路状态数据库，可以观察到数据库的变化。

`make all` 后按照实验说明操作xterm，即可重复以上列出的结果。

4. 实验的关键思路

4.1 OSPF包的构造

OSPF包分为两种: Hello和LSU，构造的方法是先新建一个char*，然后再强制类型转换为OSPF包的结构，然后往里面填空。

Hello包的构造比较简单，但LSU的包需要先计算好总共有多少个邻居(即LSA的个数)，提前malloc好，再往里面填空。

```
char* sending_packet = malloc(ETHER_HDR_SIZE + IP_BASE_HDR_SIZE +
MOSPF_HDR_SIZE + MOSPF_LSU_SIZE + nbr_count * MOSPF_LSA_SIZE);
```

```
int nbr_index = 0;
    list_for_each_entry(current_iface, &instance->iface_list, list){
        nbr_entry = NULL;
        if(current_iface->nbr_list.next != &current_iface->nbr_list){  //means
it is not empty
            list_for_each_entry_safe(nbr_entry, nbr_q, &(current_iface-
>nbr_list), list){
                struct mospf_lsa * current_mospf_lsu = (struct mospf_lsa *)
(sending_packet + ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + MOSPF_HDR_SIZE +
MOSPF_LSU_SIZE + nbr_index * MOSPF_LSA_SIZE);

                current_mospf_lsu->subnet = htonl(nbr_entry->nbr_ip &
nbr_entry->nbr_mask);
                current_mospf_lsu->mask = htonl(nbr_entry->nbr_mask);
                current_mospf_lsu->rid = htonl(nbr_entry->nbr_id);  //new
edited

                memcpy(sending_packet + ETHER_HDR_SIZE + IP_BASE_HDR_SIZE +
MOSPF_HDR_SIZE + MOSPF_LSU_SIZE + nbr_index * MOSPF_LSA_SIZE,
current_mospf_lsu, MOSPF_LSA_SIZE);
                nbr_index += 1;
            }
        }
    }
```

根据上一次router实验的经验，我们应该在最后计算checksum

```
list_for_each_entry(current_iface, &instance->iface_list, list){
        nbr_entry = NULL;
        if(current_iface->nbr_list.next != &current_iface->nbr_list){  //means
it is not empty
            list_for_each_entry_safe(nbr_entry, nbr_q, &(current_iface-
>nbr_list), list){
                memcpy(out_ether_hdr->ether_shost, current_iface->mac,
ETH_ALEN);
                out_ip_hdr->saddr = htonl(current_iface->ip);
                out_ip_hdr->daddr = htonl(nbr_entry->nbr_ip);
                instance->sequence_num += 1;
                out_mospf_lsu->seq = htons(instance->sequence_num);
                out_ospf_hdr->checksum = mospf_checksum(out_ospf_hdr);
                out_ip_hdr->checksum = ip_checksum(out_ip_hdr);
                ip_send_packet(sending_packet, ETHER_HDR_SIZE +
IP_BASE_HDR_SIZE + MOSPF_HDR_SIZE + MOSPF_LSU_SIZE + nbr_count *
MOSPF_LSA_SIZE);
            }
        }
    }
```

4.2 OSPF Hello

发送Hello的地方非常单一：对于每个路由器，每隔5秒发送一次。

4.3 OSPF LSU

发送LSU的地方比较多，我们逐一分析：

4.3.1 neighbor检查超时

当一个neighbor的alive时间超时后，会在nbr_list 中删去，这个时候会发送LSU

```
if(now - nbr_entry->alive > MOSPF_NEIGHBOR_TIMEOUT){
                        list_delete_entry(&(nbr_entry->list));
                        send_mospf_lsu();
                        printf("Time out! cleaning neighbor\n");
                    }
```

4.3.2 neighbor新增

收到hello消息，发现nbr_list中没有这个信息，把它插入到nbr_list，这时会发送LSU

```
if (found == 0){
            mospf_nbr_t * new_nbr = malloc(sizeof(mospf_nbr_t));
            new_nbr->nbr_id = ntohl(in_ospf_hdr->rid);
            new_nbr->nbr_ip = ntohl(in_ip_hdr->saddr);
            new_nbr->nbr_mask = ntohl(in_ospf_hello->mask);
            time_t now;
            now = time(NULL) - 1500000000;
            new_nbr->alive = now;
            memcpy(new_nbr->mac, in_ether_hdr->ether_shost, ETH_ALEN);

            list_add_tail(&(new_nbr->list), &(iface->nbr_list));
            send_mospf_lsu();
        }
```

4.3.3 周期性的发送LSU

```
void *sending_mospf_lsu_thread(void *param)
{
    fprintf(stdout, "TODO: send mOSPF LSU message periodically.\n");
    while(1){
        send_mospf_lsu();
        //sleep(MOSPF_DEFAULT_LSUINT);  // seems too long
        sleep(10);
    }
    return NULL;
}
```

4.4 转发收到的LSU

当一个router收到一个LSU信息后，它有可能会把这条LSU信息转发走。

在这里我们的转发条件是：

1.转发的目的IP 不等于 收到的LSU包的源IP

2.转发的目的router ID 不等于 收到的LSU包中的源router ID

代码如下：

```
list_for_each_entry(current_iface, &instance->iface_list, list){
        mospf_nbr_t *nbr_entry = NULL, *nbr_q;
        if(current_iface->nbr_list.next != &current_iface->nbr_list){  //means
it is not empty
            list_for_each_entry_safe(nbr_entry, nbr_q, &(current_iface-
>nbr_list), list){
                //if(nbr_entry->nbr_id != ntohl(in_ospf_hdr->rid)){  //can
forward
                if(nbr_entry->nbr_ip != ntohl(in_ip_hdr->saddr) && nbr_entry-
>nbr_id != ntohl(in_ospf_hdr->rid)){  //can forward
                    char * sending_packet = malloc(len);
                    memcpy(sending_packet, packet, len);
                    struct ether_header * out_ether_hdr = (struct ether_header
*)sending_packet;
                    struct iphdr * out_ip_hdr = (struct iphdr *)
(sending_packet + ETHER_HDR_SIZE);
                    struct mospf_hdr * out_ospf_hdr = (struct mospf_hdr *)
(sending_packet + ETHER_HDR_SIZE + IP_BASE_HDR_SIZE);
                    memcpy(out_ether_hdr->ether_shost, current_iface->mac,
ETH_ALEN);
                    out_ip_hdr->saddr = htonl(current_iface->ip);
                    out_ip_hdr->daddr = htonl(nbr_entry->nbr_ip);
                    out_ospf_hdr->checksum = mospf_checksum(out_ospf_hdr);
                    out_ip_hdr->checksum = ip_checksum(out_ip_hdr);
                    ip_send_packet(sending_packet, len);
                }
            }
        }
    }
```

我们进一步举个例子，就在老师给的topo里，假设从r1发出了一个LSU包，它会沿着 `r1->r2->r4->r3` 以及 `r1->r3->r4->r2` 的路径传播，在这两条路径里的最后一个节点(分别为r3 和 r2)会发现，它的下一个节点的router ID即r1的router ID，和这个LSU包的router ID是一样的！所以就停止了传播。

5.实验收获

这次实验最大的收获是我对我们的实验环境有了更深的理解。

之前对于xterm，我只知道我打开了一个东西，并且能在上面运行一个功能，而并不是很清楚是做什么的。

而本次实验中我对它的理解有了质的改变：

每一个xterm，其实就是一个节点，而这个节点是什么，则是由这个节点所运行的程序决定的。

之前我们做过switch, 做过hub，运行了这两个，就分别是交换机和集线器。

而这次实验我们做的是router，开了4个xterm，就是运行了4个router，它们之间的关联是因为我们的网络拓扑(topo.py)才有的。

而每一个router则是有不同的接口(interface)，在本次实验中，每个interface会有1或多个邻居，而这些邻居其实也是interface.

另外的收获是读了老师的lua代码，发现竟然可以配置我们自己的wireshark插件，很神奇！