《计算机网络协议开发》实验报告

第五次实验：重叠网络协议栈开发

姓名：吕玉龙

学号：191220076

2019级计算机科学与技术院/系

邮箱：1931015836@qq.com

时间：2022.5.16

一、实验目的

熟悉传输控制层信号协议的设计、实现。熟悉传输控制层协议是如何建立连接，以及如何解决信号数据包的丢失和损坏。

二、成品功能

完成了 pdf 中 5.4 所要求的所有基本功能，加上完成动态路由功能，即简单网络协议 sip 实现，可以完成验收展示。

三、实验内容

1)STCP

对 client 和 server 中需要使用上之前一直没有用上的 server_nodeid 以及 client_nodeid，所以 stcp 层也需要接用 topology，得到自身的节点 id。对于 stcp_server，需要在收到 SYN 时候，将 client 的 nodeid 存储到自己的 tcb 块中。

```
    break;
case LISTENING:
    if(packet.header.type == SYN){
        if(ServerTCB[tcbnum].client_nodeID != -1 && ServerTCB[tcbnum].client_nodeID != srcnode){
            printf("wrong receive\n");
        }
        else{
        printf("SYN received\n");
        ServerTCB[tcbnum].state = CONNECTED;
        ServerTCB[tcbnum].client_portNum = packet.header.src_port;
        if(ServerTCB[tcbnum].client_nodeID == -1)
            ServerTCB[tcbnum].client_nodeID = srcnode;
        seg_t SYNACKpacket;
        memset(&SYNACKpacket, 0 , sizeof(seg_t));
        SYNACKpacket.header.src_port = ServerTCB[tcbnum].server_portNum;
        SYNACKpacket.header.dest_port = ServerTCB[tcbnum].client_portNum;
        SYNACKpacket.header.type = SYNACK;
        SYNACKpacket.header.length = 0;
        printf("sending SYNACK packet\n");
        sip_sendseg(sip_conn,ServerTCB[tcbnum].client_nodeID,&SYNACKpacket);
        }
    }
    else{
        printf("server in LISTENING, no SYN receive\n");
    }
```

2)SIP

邻居代价表，距离矢量表，路由表都需要 son 的邻居表。但是 son 的邻居表没有办法给 sip 层借用，所以只好自己将 son 中邻居表的实现加以修改，放到了 sip.c 文件中。当然，这里需要记录两个之间的距离。出了邻居表之外，还需要记录拓扑中所有节点。

```c
typedef struct sip_neighborentry {
  int nodeID;          //邻居的节点ID
  char nodeIP[20];     //邻居的IP地址
  int conn;            //针对这个邻居的TCP连接套接字描述符
  int dis;
}sip_nbr_entry_t;

typedef struct allnode_{
  char name[20];
  int nodeID;
  struct allnode_* next;
}allnode;

allnode* mynodetable;
int all_num;
int sip_neighbor_num;
sip_nbr_entry_t* sip_nt;
char sip_myip[20];
int sip_myid;
//SIP进程使用这个函数连接到本地SON进程的端口SON_PORT
```

添加了 allnode 数据结构，用来解析拓扑中的数据结构，并加以存储，在距离矢量表中使用。自身的 ip，id，邻居数量，所有节点数量，都是必要数据，当然获取他们也很简单。

邻居代价表的初始化很简单，使用一开始的邻居表对它进行初始化即可。

```c
nbr_cost_entry_t* nbrcosttable_create()
{
  nbr_cost_entry_t* mynbrcosttable = (nbr_cost_entry_t*)malloc(sizeof(nbr_cost_entry_t)*sip_neighbor_num);
  for(int i=0;i<sip_neighbor_num;i++){
    mynbrcosttable[i].cost = sip_nt[i].dis;
    mynbrcosttable[i].nodeID = sip_nt[i].nodeID;
  }
  return mynbrcosttable;
}

//这个函数删除邻居代价表.
```

距离矢量表的初始化相对困难一些。
第一行始终存储自己，然后每行的 entry，根据 allnode 数据结构中存储的节点 id 进行分别的初始化，当然如果在邻居表中的节点，cost 都设置成对应的代价，不相邻的都设为 INFINITE_COST,其它所有行，初始化 cost 都是 INFINITE_COST，代码有点多，这里不贴图。

路由表首先需要一个哈希函数，这里直接使用了取模的哈希函数，取一个模质数的余数作为 key，随后进行初始化

```c
//该函数返回动态创建的路由表结构.
routingtable_t* routingtable_create()
{
  routingtable_t* myroutingtable = (routingtable_t*)malloc(sizeof(routingtable_t));
  for(int i = 0;i < MAX_ROUTINGTABLE_SLOTS;i++){
    myroutingtable->hash[i] = NULL;
  }
  for(int i = 0;i < sip_neighbor_num;i++){
    int key = makehash(sip_nt[i].nodeID);
    routingtable_entry_t* newnode = (routingtable_entry_t*)malloc(sizeof(routingtable_entry_t)
    newnode->destNodeID = sip_nt[i].nodeID;
    newnode->nextNodeID = sip_nt[i].nodeID;
    newnode->next = myroutingtable->hash[i];
    myroutingtable->hash[key] = newnode;
  }
  return myroutingtable;
}
```

全部初始化完成之后，就打印出它们初始化的表，随后等待路由表的更新。

3)路由广播

广播的更新报文如下，先构造广播数据报文

```
sip_pkt_t pkt;
memset(&pkt,0,sizeof(sip_pkt_t));
pkt.header.dest_nodeID = BROADCAST_NODEID;
pkt.header.type = ROUTE_UPDATE;
pkt.header.src_nodeID = sip_myid;
pkt_routeupdate_t routepkt;
memset(&routepkt,0,sizeof(pkt_routeupdate_t));
routepkt.entryNum = all_num;
pthread_mutex_lock(dv_mutex);
for(int i = 0;i < routepkt.entryNum;i++){
    routepkt.entry[i].nodeID = dv[0].dvEntry[i].nodeID;
    routepkt.entry[i].cost = dv[0].dvEntry[i].cost;
}
pthread_mutex_unlock(dv_mutex);
memcpy(pkt.data,&routepkt,sizeof(routepkt));
pkt.header.length = sizeof(routepkt);
if(son_sendpkt(BROADCAST_NODEID,&pkt,son_conn) > 0){
    printf("sending broadcast packet\n");
}
int count = 0;
```

随后判断是否到达发送时间间隔，需要重发。等待 30s，因为这里不需要 60s，为了加快实验进度，所以改成 30s，待路由表稳定后，开始进行测试。

4)pkthandler

如果收到的是路由更新报文，那么按照实验手册给出的算法进行更新。先将收到报文的 data 更新到 dvtable 中，随后，对第一行中，自身节点到其它所有节点的距离进行更新，如果有更新，就更新路由表，没有更新就不用更新路由表。这里展示部分代码

```
while(son_recvpkt(&pkt,son_conn)>0) {

    printf("Routing: received a packet from neighbor %d\n",pkt.header.src_nodeID);
    assert(pkt.header.dest_nodeID != 0);
    if(pkt.header.type == ROUTE_UPDATE){
        pthread_mutex_lock(dv_mutex);
        for(int i=0;i<sip_neighbor_num + 1;i++){
            if(dv[i].nodeID == pkt.header.src_nodeID){
                pkt_routeupdate_t recvpkt;
                memcpy(&recvpkt,pkt.data,sizeof(recvpkt));
                for(int j = 0;j<recvpkt.entryNum;j++){
                    for(int k = 0;k<recvpkt.entryNum;k++){
                        if(dv[i].dvEntry[j].nodeID == recvpkt.entry[k].nodeID){
                            dv[i].dvEntry[j].cost = recvpkt.entry[k].cost;
                            break;
                        }
                    }
                }
                break;
            }
        }
        for(int i=0;i<all_num;i++){
```

如果收到的是 SIP 报文，那么就查询路由表，找到转发的下一跳，然后发送给 son 层。

曾与曾之间的通信所用的函数都和 5.1 中的函数类似，故就不再赘述。

5)动态路由

当有一个节点意外退出时，因为绑定了 CTRL+C 的信号，所以在收到信号的时候，让 son 发送一个新的 sip_pkt，这里 pkt 类型位 DISCONNECT，这是我新加的一个类型，不同于之前路由更新以及 SIP 类型。当 sip 层收到这个类型的报文，那么就进行路由表，邻居代价表，距离矢量表的更新。方式是将两点之间的 cost 设置为 INFINITE_COST。如果成功更新，那么节点中断之后，能够找到新的代价，如果已经不能发送，就会停止发送。

```
else if(pkt.header.type == DISCONNECT){
    printf("receive\n");
    int deleteid = pkt.header.src_nodeID;
    for(int i=0;i<sip_neighbor_num;i++){
        if(nct[i].nodeID == deleteid)
            nct[i].cost = INFINITE_COST;
    }
    pthread_mutex_lock(dv_mutex);
    for(int i=0;i<sip_neighbor_num+1;i++){
        if(i == 0){
            for(int j=0;j<all_num;j++){
                if(dv[i].dvEntry[j].nodeID == dv[i].nodeID)
                    dv[i].dvEntry[j].cost = 0;
                else
                    dv[i].dvEntry[j].cost = INFINITE_COST;
            }
            for(int j=0;j<all_num;j++){
                for(int k=0;k<sip_neighbor_num;k++){
                    if(dv[i].dvEntry[j].nodeID == nct[k].nodeID)
                        dv[i].dvEntry[j].cost = nct[k].cost;
                }
            }
        }
        if(dv[i].nodeID == deleteid){
            for(int j = 0;j<all_num;j++){
```

四、问题及解决方法

5.4 的问题主要出现在多个终端的测试。

1.三个表的建立，重点是 dvtable，因为循环比较复杂，所以很难定位错误。最后发现是少写了加 1

2.constant 中设置的 pkt 长度太小，导致需要发送的时间过长，server 无法在 waittime 中接收到所有的报文

3.动态路由，一开始没法动态路由，因为收到 DISCONNECT 报文时，一开始存储的路径 cost 没有被删除，导致一直没有办法更新，仍然意图从错误的节点那里转发。