

# 《计算机网络协议开发》实验报告

## 第 15 次实验

### 简单重叠网络 SIP 的实现

姓名：元玉慧

学号：101220151

10 级 计算机 系 4 班

邮箱：njucsyyh@gmail.com

时间：2013/06/12

## 一、实验目的

通过本实验实现 SimpleNet 协议栈的简单 SIP 网络协议，使用测试程序测试整个 SimpleNet 协议栈。

## 二、实验设计背景

我们已经是实现了 STCP 协议，现在我们需要为 simpleNET 添加路由和包转发功能，我们将在实验中添加实现 SIP 层的类似于互联网中的 IP 协议和路由算法的包转发和路由支持功能。

SIP:

在每一个节点上都是有一个简单的重叠网络层 SON, 一个简单网络协议层 SIP, 一个简单的 TCP 层, 和一个应用层。SON 层被实现为一个成为 SON 的进程, SON 进程包含有 n+1 个线程,

下面主要分析理解 STCP,SIP,SON 3 个层次是如何协作的:

### STCP -> SIP (数据结构如下)

STCP 发送段到 SIP: Sip\_sendseg()

SIP 从 STCP 接收段 getsegToSend()

STCP 从 SIP 接收段: Sip\_rcvseg()

SIP 发送段到 STCP forwardsegToSTCP()

```
typedef struct stcp_hdr {
    unsigned int src_port;      //源端口号
    unsigned int dest_port;     //目的端口号
    unsigned int seq_num;       //序号
    unsigned int ack_num;       //确认号
    unsigned short int length;   //段数据长度
    unsigned short int type;     //段类型
    unsigned short int rcv_win;  //当前未使用
    unsigned short int checksum; //这个段的校验和
} stcp_hdr_t;
```

//段定义

```
typedef struct segment {
    stcp_hdr_t header;
    char data[MAX_SEG_LEN];
} seg_t;
```

//这是在SIP进程和STCP进程之间交换的数据结构.

//它包含一个节点ID和一个段.

//对sip\_sendseg()来说, 节点ID是段的目标节点ID.

//对sip\_rcvseg()来说, 节点ID是段的源节点ID.

```
typedef struct sendsegargument {
    int nodeID;    //节点ID
    seg_t seg;     //一个段
} sendseg_arg_t;
```

## SIP → SON

在 `sendpkt_arg_t` 结构体内 `pkt` 的 `data` 域封装了 STCP 层的报文 `send_seg_t`

```
//SIP报文格式定义
typedef struct sipheader {
    int src_nodeID;           //源节点ID
    int dest_nodeID;          //目标节点ID
    unsigned short int length; //报文中数据的长度
    unsigned short int type;   //报文类型
} sip_hdr_t;

//一条路由更新条目
typedef struct routeupdate_entry {
    unsigned int nodeID; //目标节点ID
    unsigned int cost;    //从源节点(报文首部中的)
} routeupdate_entry_t;

//路由更新报文格式
typedef struct pkt {
    unsigned int entryNum; //这个路由更新报文中包含
    routeupdate_entry_t entry[MAX_NODE_NUM];
} pkt_routeupdate_t;

// 数据结构sendpkt_arg_t用在函数son_sendpkt()中。
// son_sendpkt()由SIP进程调用，其作用是要求SON进程将报文
// SON进程和SIP进程通过一个本地TCP连接互连，在son_sendpkt
// SON进程通过调用getpktToSend()接收这个数据结构，然后SIP
typedef struct sendpktargument {
    int nextNodeID; //下一跳的节点ID
    sip_pkt_t pkt;   //要发送的报文
} sendpkt_arg_t;

//路由更新报文定义
//对于路由更新报文来说，路由更新信息存储在报文的data域中

//一条路由更新条目
typedef struct routeupdate_entry {
    unsigned int nodeID; //目标节点ID
    unsigned int cost;    //从源节点(报文首部)
} routeupdate_entry_t;
```

下面分析一个数据段是如何使用这些 API 从一个源节点到达目的节点的。

**在源节点处**，STCP 进程调用 `sip_sendseg()` 发送段到目的节点，当 STCP 进程调用 `sip_sendseg()` 时，一个包含段及其目的节点 IP 的结构体 `sendseg_arg_t` 被发送给本地的 SIP 进程，本地 SIP 进程使用 `getsegToSend()` 接收这个结构体，然后它将段封装进数据报，并使用 `son_sendpkt()` 将这个数据报发送给本地的 SON 进程，下一跳节点的 ID 通过 SIP 路由协议维护的路由表来获取，当 SIP 进程调用 `son_sendpkt()` 时，一个包含数据报及其下一跳节点 ID 的结构体 `send_arg_t` 被发送给本地 SON 进程。本地 SON 进程使用 `getpktToSend()` 接收这个结构，然后调用 `sendpkt()` 将这个报文发送给下一跳。

**在中间节点处**，SON 进程调用 `recvpkt()` 接收报文，然后调用 `forwardpktToSIP()` 将报文转发给本地 SIP 进程，本地 SIP 进程调用 `son_recvpkt()` 接收由 SON 进程转发的报文，然后通过路由表获取下一跳节点的 ID，并调用 `son_sendpkt()` 发送 `send_arg_t` 给本地的 SON 进程，本地 Son 进程使用 `getpktToSend()` 接收这个结构，然后调用 `sendpkt()` 发送报文给下一跳的节点。

**在目的节点处 SON 进程**，调用 `recvpkt()` 从邻居节点处接收报文，然后调用 `forwardToSIP()` 转发给本地 SIP 进程，本地 SIP 进程调用 `son_recvpkt()` 接收又 SON 进程转发的报文，然后从报文的 `data` 字段中提取段，并将包含段的源节点 ID 和段本身的 `sendseg_arg_t` 结构发送给本地的 STCP 进程，本地 STCP 进程调用 `sip_recvseg()` 接收这个结构。

下面主要分析 SIP API 的实现：

SIP 进程为 STCP 进程提供了 2 个函数，`sip_sendseg()` 和 `sip_recvseg()`，但是我们之前的实现是同过 2 个节点之间直接的 TCP 连接实现的，本实验中，我们要基于上一个实验中的 SON 层来重新实现这 2 个函数。

STCP 进程调用 sip\_sendseg() 发送段到目的节点，当 STCP 进程调用 sip\_sendseg() 时，一个包含段及其目的节点 ID 的结构 send\_arg\_t 被发送给本地的 SIP 进程，本地 SIP 进程使用 getsetToSend() 接收这个结构，然后将它封装进数据包，并使用 son\_sendpkt() 将这个数据报发送个下一跳，下一跳节点的 ID 通过 SIP 路由协议维护的路由表来获取。

SIP 进程调用 son\_rcvpkt() 接收来自本地 SON 进程的报文，然后从报文的 data 字段中提取段，并将包含段的源节点和段本身的结构体转发给本地 STCP 进程。本地 STCP 进程调用 sip\_rcvseg() 接收这个结构。

Send\_arg\_t 结构定义在头文件 seg.h 中：

SIP 路由协议实现部分：

SIP 路由协议使用距离矢量路由算法，维护使用 3 个表，邻居代价表，距离矢量表，路由表；

每个节点都维护一个 SIP 进程，每一个 SIP 进程为运行该进程的节点维护 3 个表。

接收者的距离矢量表和路由表的更新是通过 2 个步骤完成：

假设路由更新报文的源节点是 S，接收者是 X，

**步骤 1：** X 使用路由更新报文中包含的距离矢量更新 S 的距离矢量，X 的距离矢量

**步骤 2：** X 的距离矢量被重新计算，X 的路由表给更新，为重新计算 X 的距离矢量，针对每个目的节点有，从 X 到 Y 的新的估计链路代价……

SIP 进程启动，它首先创建一个邻居代价表，一个距离矢量表和一个路由表，并初始化这 3 个表，SIP 进程还创建并初始化 2 个互斥变量，一个用于距离实现表，一个用于路由表，SIP 进程维护了一个 TCP 描述符，son\_conn 和一个 TCP 描述符 stcp\_conn，当 SIP 进程启动时，都初始化为 -1；

### 三、实验实现

邻居代价表函数实现： /nbrcosttable.c

```

21 routingtable_t* routingtable_create()
22 {
23     int i;
24     int neighbor_count = topology_getNbrNum();
25     int *neighbor_list = topology_getNbrArray();
26     routingtable_t* list = (routingtable_t*)malloc(sizeof(routingtable_t));
27     //initiate the hash list..
28     //hash is the head of the list
29     for(i=0; i<MAX_ROUTINGTABLE_SLOTS; i++){
30         list->hash[i] = (routingtable_entry_t *)malloc(sizeof(routingtable_entry_t));
31         list->hash[i]->destNodeID = -1;
32         list->hash[i]->nextNodeID = -1;
33         list->hash[i]->next = NULL;
34     }
35     for(i=0; i<neighbor_count; i++){
36         int hash = makehash(neighbor_list[i]);
37         routingtable_entry_t* new_node = (routingtable_entry_t *)malloc(sizeof(routingtable_entry_t));
38         new_node->destNodeID = neighbor_list[i];
39         new_node->nextNodeID = neighbor_list[i];
40         new_node->next = NULL;
41         //insert the hash node at the head
42         if(list->hash[hash]->next == NULL)
43             list->hash[hash]->next = new_node;
44         else{
45             new_node->next = list->hash[hash]->next;
46             list->hash[hash]->next = new_node;
47         }
48     }
49     return list;
50 }

```

```

54 void routingtable_destroy(routingtable_t* routingtable)
55 {
56     int i;
57     for(i=0; i<MAX_ROUTINGTABLE_SLOTS; i++){
58         while(routingtable->hash[i]->next != NULL){
59             routingtable_entry_t* temp = routingtable->hash[i]->next;
60             routingtable->hash[i]->next = temp->next;
61             free(temp);
62         }
63     }
64     free(routingtable);
65     return;
66 }

```

```

74 void routingtable_setnextnode(routingtable_t* routingtable, int destNodeID, int nextNodeID)
75 {
76     int hash = makehash(destNodeID);
77     routingtable_entry_t* cur = routingtable->hash[hash]->next;
78     while(cur != NULL){
79         if(cur->destNodeID == destNodeID){
80             cur->nextNodeID = nextNodeID;
81             break;
82         }
83         cur = cur->next;
84     }
85     //insert at the head
86     if(cur == NULL){
87         routingtable_entry_t* new_node = (routingtable_entry_t *)malloc(sizeof(routingtable_entry_t));
88         new_node->destNodeID = destNodeID;
89         new_node->nextNodeID = nextNodeID;
90         new_node->next = routingtable->hash[hash]->next;
91         routingtable->hash[hash]->next = new_node;
92     }
93     return;
94 }

```

距离矢量表函数实现： /dvtable.c

```
18 dv_t* dvtable_create()
19 {
20     int hostID = topology_getMyNodeID();
21     int neighbor_count = topology_getNbrNum();
22     int all_count = topology_getNodeNum();
23     printf("all_count is %d\n", all_count);
24     dv_t *dis_vector = (dv_t *)malloc((neighbor_count+1)*sizeof(dv_t));
25     //the neighbor_list store the neighbor ID
26     //the all_list store all the ID
27     int *neighbor_list, *all_list;
28     int i, j;
29     neighbor_list = topology_getNbrArray();
30     all_list = topology_getNodeArray();
31     //store the local distance_vector
32     dis_vector[0].dvEntry = (dv_entry_t*)malloc(all_count * sizeof(dv_entry_t));
33     for(i=0; i<all_count; i++){
34         dis_vector[0].nodeID = hostID;
35         dis_vector[0].dvEntry[i].nodeID = all_list[i];
36         dis_vector[0].dvEntry[i].cost = topology_getCost(hostID, all_list[i]);
37     }
38     //init the srcID
39     //alloc space for the dv_Entry
40     for(i=0; i<neighbor_count; i++){
41         dis_vector[i+1].nodeID = neighbor_list[i];
42         dis_vector[i+1].dvEntry = (dv_entry_t*)malloc(all_count * sizeof(dv_entry_t));
43         for(j=0; j<all_count; j++){
44             dis_vector[i+1].dvEntry[j].nodeID = all_list[j];
45             dis_vector[i+1].dvEntry[j].cost = INFINITE_COST;
46         }
47     }
48     return dis_vector;
49 }

66 int dvtable_setcost(dv_t* dvtable, int fromNodeID, int toNodeID, unsigned int cost)
67 {
68     int neighbor_count = topology_getNbrNum();
69     int all_count = topology_getNodeNum();
70     int i, j;
71     for(i=0; i<neighbor_count+1; i++){
72         if(dvtable[i].nodeID == fromNodeID) break;
73     }
74     for(j=0; j<all_count; j++){
75         if(dvtable[i].dvEntry[j].nodeID == toNodeID){
76             dvtable[i].dvEntry[j].cost = cost;
77             return 1;
78         }
79     }
80     return -1;
81 }
```

```

85 unsigned int dvtable_getcost(dv_t* dvtable, int fromNodeID, int toNodeID)
86 {
87     int neighbor_count = topology_getNbrNum();
88     int all_count = topology_getNodeNum();
89     int i, j;
90     for(i=0; i<neighbor_count+1; i++){
91         if(dvtable[i].nodeID == fromNodeID) break;
92     }
93     for(j=0; j<all_count; i++){
94         if(dvtable[i].dvEntry[j].nodeID == toNodeID){
95             return dvtable[i].dvEntry[j].cost;
96         }
97     }
98     return INFINITE_COST;
99 }

```

路由表函数实现： /routingtable.c

```

21 routingtable_t* routingtable_create()
22 {
23     int i;
24     int neighbor_count = topology_getNbrNum();
25     int *neighbor_list = topology_getNbrArray();
26     routingtable_t* list = (routingtable_t*)malloc(sizeof(routingtable_t));
27     //initiate the hash list..
28     //hash is the head of the list
29     for(i=0; i<MAX_ROUTINGTABLE_SLOTS; i++){
30         list->hash[i] = (routingtable_entry_t *)malloc(sizeof(routingtable_entry_t));
31         list->hash[i]->destNodeID = -1;
32         list->hash[i]->nextNodeID = -1;
33         list->hash[i]->next = NULL;
34     }
35     for(i=0; i<neighbor_count; i++){
36         int hash = makehash(neighbor_list[i]);
37         routingtable_entry_t* new_node = (routingtable_entry_t *)malloc(sizeof(routingtable_entry_t));
38         new_node->destNodeID = neighbor_list[i];
39         new_node->nextNodeID = neighbor_list[i];
40         new_node->next = NULL;
41         //insert the hash node at the head
42         if(list->hash[hash]->next == NULL)
43             list->hash[hash]->next = new_node;
44         else{
45             new_node->next = list->hash[hash]->next;
46             list->hash[hash]->next = new_node;
47         }
48     }
49     return list;
50 }

```

```

52 //这个函数删除路由表.
53 //所有为路由表动态分配的数据结构将被释放.
54 void routetable_destroy(routetable_t* routetable)
55 {
56     int i;
57     for(i=0; i<MAX_ROUTINGTABLE_SLOTS; i++){
58         while(routetable->hash[i]->next != NULL){
59             routetable_entry_t* temp = routetable->hash[i]->next;
60             routetable->hash[i]->next = temp->next;
61             free(temp);
62         }
63     }
64     free(routetable);
65     return;
66 }

74 void routetable_setnextnode(routetable_t* routetable, int destNodeID, int nextNodeID)
75 {
76     int hash = makehash(destNodeID);
77     routetable_entry_t* cur = routetable->hash[hash]->next;
78     while(cur != NULL){
79         if(cur->destNodeID == destNodeID){
80             cur->nextNodeID = nextNodeID;
81             break;
82         }
83         cur = cur->next;
84     }
85     //insert at the head
86     if(cur == NULL){
87         routetable_entry_t* new_node = (routetable_entry_t *)malloc(sizeof(routetable_entry_t));
88         new_node->destNodeID = destNodeID;
89         new_node->nextNodeID = nextNodeID;
90         new_node->next = routetable->hash[hash]->next;
91         routetable->hash[hash]->next = new_node;
92     }
93     return;
94 }

99 int routetable_getnextnode(routetable_t* routetable, int destNodeID)
100 {
101     int hash = makehash(destNodeID);
102     routetable_entry_t* cur = routetable->hash[hash]->next;
103     while(cur != NULL){
104         if(cur->destNodeID == destNodeID) return cur->nextNodeID;
105         cur = cur->next;
106     }
107     return -1;
108 }
109

```

Sip API 实现

-1- routeupdate\_daemon()



```

74 void* routeupdate_daemon(void* arg) {
75     int j;
76     int index = 0;
77     int hostID = topology_getMyNodeID();
78     int neighbor_count = topology_getNbrNum();
79     int all_count = topology_getNodeNum();
80     int size_of_data = (neighbor_count+1)*all_count*sizeof(int)*2 + sizeof(int);
81     sip_pkt_t temp;
82     temp.header.src_nodeID = hostID;
83     temp.header.dest_nodeID = BROADCAST_NODEID;
84     temp.header.length = size_of_data;
85     temp.header.type = ROUTE_UPDATE;
86     //fill in the info into the data field
87     pkt_routeupdate_t *data = (pkt_routeupdate_t *)malloc(sizeof(pkt_routeupdate_t));
88     data->entryNum = all_count;
89     while(1){
90         for(j=0; j<all_count; j++){
91             data->entry[index].nodeID = dv[0].dvEntry[j].nodeID;
92             data->entry[index].cost = dv[0].dvEntry[j].cost;
93             index++;
94         }
95         memcpy(temp.data, data, sizeof(pkt_routeupdate_t));
96         son_sendpkt(BROADCAST_NODEID, &temp, son_conn);
97         sleep(5);
98     }
99 }

```

-2-

主要是添加了处理来自 SON 进程的报文，需要区分处理多种情况

```

104 void* pkthandler(void* arg) {
105     sip_pkt_t pkt;
106     int hostID = topology_getMyNodeID();
107     int neighbor_count = topology_getNbrNum();
108     int all_count = topology_getNodeNum();
109     int *all_list = topology_getNodeArray();
110     int i, j;
111
112     while(son_recvpkt(&pkt, son_conn)>0) {
113         //printf("id:%d->%d port%d\n", pkt.header.src_nodeID, pkt.header.dest_nodeID, ((seg_t *) (pkt.data))>header.src_port);
114         if(pkt.header.type==SIP){
115             int srcID = pkt.header.src_nodeID;
116             int destID = pkt.header.dest_nodeID;
117             /* case 1 */
118             if(destID == hostID){
119                 printf("case1 ... \n");
120                 seg_t *seg = malloc(sizeof(seg_t));
121                 memcpy(seg, &(pkt.data), sizeof(seg_t));
122                 forwardsegToSTCP(stcp_conn, srcID, seg);
123             }
124             /* case 2 */
125             else{
126                 pthread_mutex_lock(routingtable_mutex);
127                 int nextID = routingtable_getnextnode(routingtable, destID);
128                 pthread_mutex_unlock(routingtable_mutex);
129                 if(nextID != -1) {
130                     printf("case2 ... \n");
131                     son_sendpkt(nextID, &pkt, son_conn);
132                 }
133                 else {
134                     printf("fail to get the nextID info ... \n");
135                 }
136             }
137         }
138     }
139 }

```

图上打五角星部分是路由更新的计算部分代码：


```

/* case 3 */
//route update info
else{
    int srcID = pkt.header.src_nodeID;
    //lock the cri...
    pthread_mutex_lock(dv_mutex);
    pthread_mutex_lock(routingtable_mutex);
    //update the distance_vector
    //part 0
    /* update the line that the srcnode ID is the srcID */
    for(i=0; i<neighbor_count+1; i++){
        if(dv[i].nodeID == srcID) break;
    }

    for(j=0; j<all_count; j++){
        dv[i].dvEntry[j].cost = ((pkt_routeupdate_t *)(&(pkt.data)))->entry[j].cost;
    }
    //part 1
    /* update the line 0 that the srcnode ID is the hostID */
    /* fetch the current distance of the srcID and the hostID */

    int temp_distance = nbrcosttable_getcost(nct, srcID);
    //part 2
    /* update the distance of the line 0 by compare the distance through srcID */
    for(i=0; i<all_count; i++){
        if(dv[0].dvEntry[i].cost > (temp_distance + ((pkt_routeupdate_t *)(&(pkt.data)))->entry[i].cost)){
            //update the distance
            dv[0].dvEntry[i].cost = temp_distance + ((pkt_routeupdate_t *)(&(pkt.data)))->entry[i].cost;
            //update the routing table
            routingtable_setnextnode(routingtable, all_list[i], srcID);
        }
    }
    //unlock the cri...
    pthread_mutex_unlock(dv_mutex);
}

```



-3- 等待 STCP 接入的函数

```

195 void waitSTCP() {
196     int hostID = topology_getMyNodeID();
197     socklen_t cliilen;
198     int listenfd;
199     //connfd = 0;
200     struct sockaddr_in cliaddr, servaddr;
201
202     if((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
203         perror("Problem in creating the server socket...\n");
204         exit(2);
205     }
206     servaddr.sin_family = AF_INET;
207     servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
208     servaddr.sin_port = htons(SIP_PORT);
209
210     bind(listenfd, (struct sockaddr *)&servaddr, sizeof(servaddr));
211     listen(listenfd, 20);
212     printf("SIP level : Server running ... wait for connections ... \n");
213     //当本地STCP进程断开连接时, 这个函数等待下一个STCP进程的连接.
}

```

```

214 while(1){
215     clilen = sizeof(cliaddr);
216     stcp_conn = accept(listenfd, (struct sockaddr *)&cliaddr, &clilen);
217     int destID, nextID;
218     seg_t segPtr;
219     sip_pkt_t pkt;
220     while(getsegToSend(stcp_conn, &destID, &segPtr) > 0){
221         /*
222          * fix a bug...
223          */
224         pkt.header.src_nodeID = hostID;
225         pkt.header.dest_nodeID = destID;
226         pkt.header.length = sizeof(seg_t);
227         pkt.header.type = SIP;
228         memcpy(pkt.data, &segPtr, sizeof(seg_t));
229         //check the routing table
230         pthread_mutex_lock(routingtable_mutex);
231         nextID = routingtable_getnextnode(routingtable, destID);
232         pthread_mutex_unlock(routingtable_mutex);
233         son_sendpkt(nextID, &pkt, son_conn);
234     }
235     close(stcp_conn);
236 }
237 close(listenfd);
238 return;
239 }

```

SON 层函数的修改

写程序的时候遇到的 BUG 主要出现在 waitSIP()函数内部,

```

while(1){
    sip_pkt_t pkt;
    int nextNode;
    getpktToSend(&pkt, &nextNode, sip_conn);
    int Count = topology_getNbrNum();
    int i;
    /* case 1 */
    if(nextNode == BROADCAST_NODEID){
        for(i=0; i<Count; i++){
            sendpkt(&pkt, nt[i].conn);
        }
    }
    /* case 2 */
    else{
        /*
         *fix the bug !!!
         */
        for(i=0; i<Count; i++){
            if(nt[i].nodeID == nextNode) break;
        }
        sendpkt(&pkt, nt[i].conn);
    }
}
}

```

当时调试程序的时候遇到的问题就是上面这段代码实现的有问题，就是路由更新报文可以正常的处理发送，但是发送的 SIP 报文不能得到正常的处理，后来发现问题是我处理 case2 的时候，没有去邻居表里面查找，而是直接发送给对应的 nextNode，参数本应是套接字描述符，我传进去的是下一跳的 ID.所以 SIP 报文本身就没有被正常的发送出去。

以上是个人认为较重要的部分和个人在实现时出错的地方，其余代码实现部分见代码。

## 四、实验结果

基于 sip, son, stcp 协议栈我们实现了在自己的协议栈上收发文件，如下分别对 simple 和 stress 的情况进行了测试：

首先是运行 son 层的进程：

```
114.212.190.185 - default - SSH Secure...
File Edit View Window Help
Quick Connect Profiles
***** END *****
Overlay network: neighbor 1:187
Overlay network: neighbor 2:186
Overlay network: neighbor 3:188
SON level : Server running ... wait for connections ...
SON level: Receive the request...5
SON level: Receive the request...6
SON level: Receive the request...7
Overlay network: node initialized...
Overlay network: waiting for connection
Connected to 114.212.190.185 SSH2 - aes128-

114.212.190.186 - default - SSH Secure ...
File Edit View Window Help
Quick Connect Profiles
nodeID: 185 -- nodeIP: 3116291186 -- conn:-1
***** END *****
Overlay network: neighbor 1:188
Overlay network: neighbor 2:185
SON level : Server running ... wait for connections ...
SON level: Receive the request...5
Overlay network: node initialized...
Overlay network: waiting for connection from SIP process...
debug : waitSIP()..
Connected to 114.212.190.186 SSH2 - aes128-c

114.212.190.187 - default - SSH Secure ...
File Edit View Window Help
Quick Connect Profiles
conn:-1
***** END *****
Overlay network: neighbor 1:188
Overlay network: neighbor 2:185
SON level : Server running ... wait for connections ...
SON level: Receive the request...5
creat a listening thread ...
Overlay network: node initialized...
Overlay network: waiting for connection from SIP process...
Connected to 114.212.190.187 SSH2 - aes128-c

114.212.190.188 - default - SSH Secure ...
File Edit View Window Help
Quick Connect Profiles
***** END *****
Overlay network: neighbor 1:187
Overlay network: neighbor 2:186
Overlay network: neighbor 3:185
SON level : Server running ... wait for connections ...
creat a listening thread ...
creat a listening thread ...
Overlay network: node initialized...
Overlay network: waiting for connection from SIP process...
Connected to 114.212.190.188 SSH2 - aes128-c
```

然后运行 sip 层的进程：

并且 sip 层的不同的主机之间通过收发路由更新报文，更新路由表：

185 更新后的路由表

186 更新后的路由表

```

***** Routing Table *****
***
destNodeID      nextNodeID
186      186
187      187
188      187
*****      END      *****
waiting for connection from STCP process
SIP level : Server running ... wait for connections ...

```

---

187 更新后的路由表

```

***** Routing Table *****
***
destNodeID      nextNodeID
185      185
186      188
188      188
*****      END      *****
waiting for connection from STCP process
SIP level : Server running ... wait for connections ...

```

```

***** Routing Table *****
**
destNodeID      nextNodeID
185      185
187      188
188      188
*****      END      *****
waiting for connection from STCP process
SIP level : Server running ... wait for connections ...
-----1

```

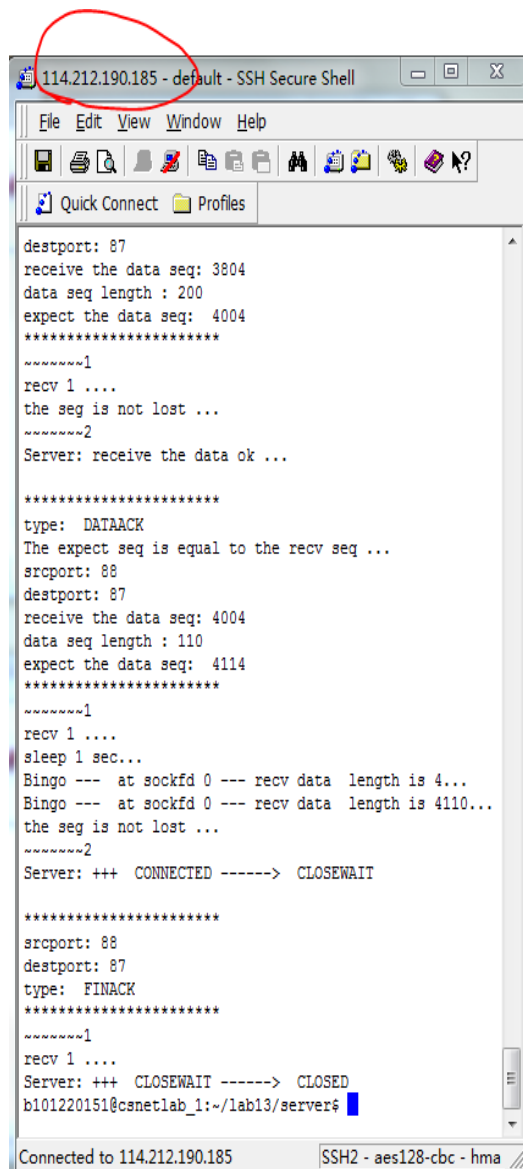
更新后的路由表 188

```

***** Routing Table *****
***
destNodeID      nextNodeID
185      187
186      186
187      187
*****      END      *****
waiting for connection from STCP process
SIP level : Server running ... wait for connections ...

```

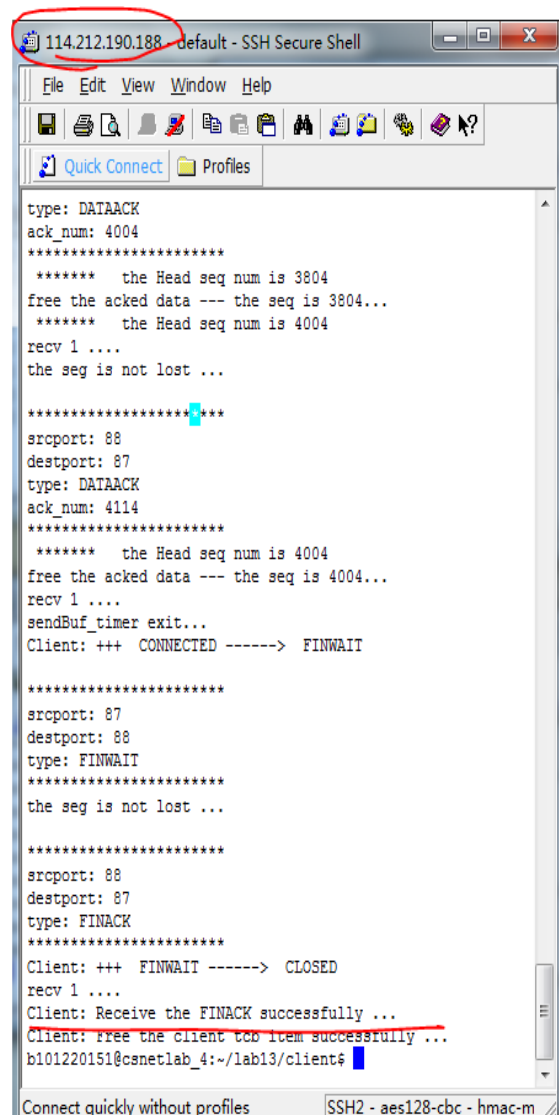
测试 185 和 188 之间的压力测试:

A screenshot of an SSH terminal window titled "114.212.190.185 - default - SSH Secure Shell". The window shows a sequence of network-related messages. It starts with "destport: 87", "receive the data seq: 3804", "data seq length : 200", and "expect the data seq: 4004". There are several lines of asterisks and "~~~~~" indicating a sequence of events. It then shows "recv 1 ....", "the seg is not lost ...", and "Server: receive the data ok ...". This is followed by "type: DATAACK", "The expect seq is equal to the rcv seq ...", "srcport: 88", "destport: 87", "receive the data seq: 4004", "data seq length : 110", and "expect the data seq: 4114". More asterisks and "~~~~~" follow. Then "recv 1 ....", "sleep 1 sec...", "Bingo --- at sockfd 0 --- rcv data length is 4...", "Bingo --- at sockfd 0 --- rcv data length is 4110...", "the seg is not lost ...", and "Server: +++ CONNECTED -----> CLOSEWAIT". This is followed by "srcport: 88", "destport: 87", "type: FINACK", and "Client: Receive the FINACK successfully ...". The final line shows the user prompt "b101220151@csnetlab\_1:~/lab13/server\$". The status bar at the bottom indicates "Connected to 114.212.190.185" and "SSH2 - aes128-cbc - hmac".

```
destport: 87
receive the data seq: 3804
data seq length : 200
expect the data seq: 4004
*****
~~~~~1
recv 1 ....
the seg is not lost ...
~~~~~2
Server: receive the data ok ...

*****
type: DATAACK
The expect seq is equal to the rcv seq ...
srcport: 88
destport: 87
receive the data seq: 4004
data seq length : 110
expect the data seq: 4114
*****
~~~~~1
recv 1 ....
sleep 1 sec...
Bingo --- at sockfd 0 --- rcv data length is 4...
Bingo --- at sockfd 0 --- rcv data length is 4110...
the seg is not lost ...
~~~~~2
Server: +++ CONNECTED -----> CLOSEWAIT

*****
srcport: 88
destport: 87
type: FINACK
*****
~~~~~1
recv 1 ....
Server: +++ CLOSEWAIT -----> CLOSED
b101220151@csnetlab_1:~/lab13/server$
```

A screenshot of an SSH terminal window titled "114.212.190.188 - default - SSH Secure Shell". The window shows a sequence of network-related messages. It starts with "type: DATAACK", "ack\_num: 4004", and several lines of asterisks. Then "\*\*\*\*\* the Head seq num is 3804", "free the acked data --- the seq is 3804...", "\*\*\*\*\* the Head seq num is 4004", "recv 1 ....", "the seg is not lost ...", and "Client: +++ CONNECTED -----> FINWAIT". This is followed by "srcport: 88", "destport: 87", "type: DATAACK", "ack\_num: 4114", and several lines of asterisks. Then "\*\*\*\*\* the Head seq num is 4004", "free the acked data --- the seq is 4004...", "recv 1 ....", "sendBuf\_timer exit...", "Client: +++ CONNECTED -----> FINWAIT". This is followed by "srcport: 87", "destport: 88", "type: FINWAIT", and several lines of asterisks. Then "the seg is not lost ...", "srcport: 88", "destport: 87", "type: FINACK", and several lines of asterisks. Then "Client: +++ FINWAIT -----> CLOSED", "recv 1 ....", "Client: Receive the FINACK successfully ...", and "Client: free the client tcb item successfully ...". The final line shows the user prompt "b101220151@csnetlab\_4:~/lab13/client\$". The status bar at the bottom indicates "Connect quickly without profiles" and "SSH2 - aes128-cbc - hmac-m".

```
type: DATAACK
ack_num: 4004
*****
***** the Head seq num is 3804
free the acked data --- the seq is 3804...
***** the Head seq num is 4004
recv 1 ....
the seg is not lost ...
Client: +++ CONNECTED -----> FINWAIT

*****
srcport: 88
destport: 87
type: DATAACK
ack_num: 4114
*****
***** the Head seq num is 4004
free the acked data --- the seq is 4004...
recv 1 ....
sendBuf_timer exit...
Client: +++ CONNECTED -----> FINWAIT

*****
srcport: 87
destport: 88
type: FINWAIT
*****
the seg is not lost ...

*****
srcport: 88
destport: 87
type: FINACK
*****
Client: +++ FINWAIT -----> CLOSED
recv 1 ....
Client: Receive the FINACK successfully ...
Client: free the client tcb item successfully ...
b101220151@csnetlab_4:~/lab13/client$
```

可以看到服务器端 185 接收的数据的长度是 4110，并且打印了成功接收数据段的信息。  
测试 OK!!!

## 五、实验中遇到的问题

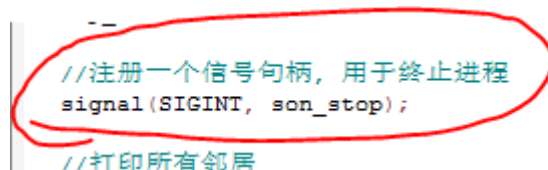
### 实验中遇到的问题：

Sip.c 中的接收线程，只能接受正常的路由更新报文，而对于后面发送的带有数据段的 SIP 报文却没有正确的接收？？？

后经调试发现了不少设计模块的问题，不过最终都解决了，部分是本来应该连接服务器的操作，写成了等待客户端，还有就是发送或者接收报文的时候，主要的 BUG，是由于代码分析部分中提到的，就是关于在处理 SIP 报文的时候发送的时候，参数传递错误。对自己实在是无语啊!!

### 程序运行到 send()函数的时候就会挂掉?

这个问题是由于 send 的时候，接收端对应的端可能已经关闭，所以系统会提供一个默认的处理操作，是直接退出程序的，所以我们需要对系统的设置进行修改，主要就是在执行 send () 之前先添加一句：



```
--  
//注册一个信号句柄，用于终止进程  
signal(SIGINT, son_stop);  
//打印所有邻居
```

需要包含头文件：#include <signal.h>

### 程序中在处理 3 个收发层次的时候逻辑不清，引起了程序收发不正确的问题？

这些 bug 都是读程序解决的，对自己写的程序从头到尾读了大概 3 遍左右，大概理清了思路，然后就把一些小 bug 都调试好了。

## 六、实验的启示/意见和建议

实验中遇到了各种 bug，不过一个学期都是这样过来的，已经有点适应了，感觉自己的 C 语言编程能力是有提升，但是代码风格还不够好，以后一定要重视代码风格的养成和训练，因为说句吐血的实话，代码风格好的程序不容易有 bug，调试起来方便，读起来也方便，而自己写的代码，实在是无语啦!!!

对我来说，我离自己的目标还很遥远!!! 但是我会持续的努力下去！终有一天我会触及到自己的理想的彼岸的！

此次代码编写花了 10 小时 调试花了 20 小时…。我已经吐血啦，实在是太花时间啦，以后一定要成为码神！