《计算机网络协议开发》实验报告

第 14 次实验
简单重叠网络 SON 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：2013/06/08

一、 实验目的

通过本实验实现 SimpleNet 协议栈中重叠网络层 SON, 使用一个简单的网络驱动程序测试重叠网络层的实现。

二、实验设计背景

我们已经是实现了 STCP 协议，现在我们开始实现协议栈中的重叠网络层。SimpleNet 的构建基于一个重叠网络，本质上，我们在终端节点上实现了 internet 路由器，然后根据路由图创建这些重叠节点之间的网络，这允许我们在重叠节点上实现 SIP, 从而绕开不能改变路由器代码的限制。

重叠网络层通过一个进程实现。简单重叠网络进程 SON

SON 进程维护该节点和所有他的邻居之间的 TCP 连接和一个 SIP 层之间的本地 TCP 连接。SON 进程为多线程，针对到每一个邻居的 TCP 连接，它都有一个 listen_to_neighbor 线程用于持续接收来自该邻居的报文，并将报文转发给 SIP 进程，对于 SON 进程与 SIP 进程之间的 TCP 连接，main 线程持续的接收来自 SIP 进程的 sendpkt_arg_t 结构，并将这些报文发送给下一跳节点。

重叠网络拓扑和节点 ID

NodeID 用来标识一个主机，其作用于 TCP/IP 中的地址类似，nodeID 用一个主机 IP 地址的最后 8 位所代表的整数来表示。

202.119.32.12 中的 12 就是 nodeID

构建重叠网络:

SON 进程的工作方式:

我们重叠网络中共有 4 个节点，

每个节点都运行一个 SON 进程和一个 SIP 进程。

SON 进程和 SIP 进程之间通过一个本地 TCP 连接。

每个 SON 进程还维护到其所有邻居节点之间的 TCP 连接。

每个 SON 进程维护了 4 条线程：main 线程+3 条 listen_to_neighbor()线程

邻居表:

运行在每个节点上的 SON 进程都维护了一个邻居表，该表包含了邻居节点信息。邻居表定义。需要通过解析文件 topology.dat 来提取信息，邻居表中的每一个条目都包含有邻居的节点 ID 邻居 IP 地址 和到该邻居的 TCP 连接的套接字描述符

（直接提取的）邻居的节点 ID 邻居 IP 地址 （在建立连接的时候添加的）到该邻居的 TCP 连接的套接字描述符

简单网络协议 SIP 报文格式

SON 进程从重叠网络中接收 SIP 报文，并将这些 SIP 报文转发给 SIP 层，SON 进程还接收来自 SIP 层的 SIP 报文，并将这些报文转发到重叠网络中。SIP 报文格式定义在头文件中。

重叠网络的控制流程

SIP 进程根据路由表，

-1- 调用 son_send_pkt()发送报文给下一跳，当 SIP 进程调用 son_sendpkt()时，一个包含下一跳节点 ID 和报文本身的 sendpkt_arg_t 结构体将会发送给本地的 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。

-2- SON 进程对到每个邻居节点的 TCP 连接都运行一个线程，在每一个线程中，recvpkt()函数用于接受来自邻居节点的报文，SON 进程则通过调用 forwardpktToSIP()将接收到的报文转发给本地的 SIP 进程。本地 SIP 进程调用 son_recvpkt()接收转发自 SON 进程的报文

-3- SON 进程和 SIP 进程通过一个本地 TCP 连接互连，重叠网络中的节点也通过 TCP 连接互连，为了通过 TCP 连接发送数据，分割符与之前相同“！&”表示开始，“！#”表示传输结束。所以通过 TCP 连接发送数据，对于端接收数据时可以考虑使用一个简单的 FSM.

SON 进程为 SIP 进程提供了 2 个函数调用：son_sendpkt()和 son_recvpkt()

当 SON 进程启动的时候，它动态创建一个邻居表，并使用来自拓扑文件的信息来初始化邻居表，SON 进程还将到 SIP 进程的连接初始化为无效值-1；（？？？） SON 进程然后为信号 SIGINT 注册信号句柄， son_stop()，这样 SON 进程接收到信号 SIGINT 时，改句柄将被调用以终止 SON 进程。

然后建立链接到重叠网络中所有邻居的 TCP 连接，邻居节点信息从文件 topology.dat 中提取，如果 2 个节点之间有一个链接，节点 ID 较小的节点将在端口 CONNECT_PORT 上监听 TCP 连接，节点 ID 较大的节点将链接到该端口。

基本运行流程

当 SON 进程启动的时候，它动态创建一个邻居表，并使用来自文件的 topology.dat 中的信息初始化邻居表，SON 进程还将到 SIP 进程的连接初始化为-1. SON 进程然后为信号 SIGINT 注册新号句柄 son_stop()，这样当 SON 进程接收到信号 SIGINT 时，该句柄被调用以终止 SON 进程。

然后建立到重得网络中所有邻居的 TCP 连接，邻居节点信息从文件 topology.dat 中提取，

SON 进程启动 waitNbrs 线程，然后经过一段时间后再调用 connectNbrs()函数，waitNbrs 线程在端口 CONNECTION_PORT 上监听 TCP 连接，等待节点 ID 比自己大的所有邻居连接到该端口。在所有进入连接建立之后，该线程就返回，主线程在启动 connectNbrs()函数返回后，主线程等待 waitNbrs 线程返回，在该线程返回后，所有与邻居之间的 TCP 连接全部都建立好。

对于连接到每个邻居的 TCP 连接，SON 进程启动一个 listen_to_neighbor 线程，每个 listen_to_neighbor 线程使用 recvpkt()从邻居接收报文，并使用 forwardToSIP 将报文转发给 SIP 进程。

一旦所有的 listen_to_...线程都启动了，SON 进程就调用 waitSIP()函数，这个函数打开 TCP 端口 SON_PORT，并等待来自本地 SIP 进程的 TCP 连接，在连接建立后，waitSIP()使用 sendpkt()将报文发送到重叠网络中的下一跳，当 SIP 进程断开连接时，waitSIP()关闭连接，并等待来自本地 SIP 进程的下一次进入连接。

三、实验内容

简单传输控制协议，STCP

-1- 首先需要实现对于每个端点处的邻居拓扑结构进行解析，并建立邻居表，其中使用到了一些系统提供的结构体和函数。

-1.1- 路由表部分

(1) 建立路由表的函数实现

```
nbr_entry_t* nt_create()
{
   char hname[HOSTNAMESIZE];
   gethostname(hname, sizeof(hname));
   printf("the host name is %s\n", hname);
   int NbrCount = 0;
   FILE *fp;
   if((fp=fopen("../topology/topology.dat","r"))==NULL) {
       printf("cannot open this file\n");
       exit(1);
   }
   NbrCount = topology_getNbrNum();
   char host[MAXHOST][20], distance[20];
   nbr_entry_t*  head;
   head = (nbr_entry_t*)malloc(NbrCount*sizeof(nbr_entry_t));
   int i=0;
   int index=0;
   while(fscanf(fp, "%s", host[i++]) > 0){
     printf("host[%d]is  %s \n",i-1, host[i-1]);
     fscanf(fp, "%s", host[i++]);
     fscanf(fp, "%s", distance);
     if(strncmp(host[i-1], hname, 20) == 0){
       head[index].nodeID = topology_getNodeIDfromname(host[i-2]);
       head[index].nodeIP = getNodeIPfromname(host[i-2]);
       head[index].conn = -1;
       index++;
     }
     if(strncmp(host[i-2], hname, 20) == 0){
       head[index].nodeID = topology_getNodeIDfromname(host[i-1]);
       head[index].nodeIP = getNodeIPfromname(host[i-1]);
       head[index].conn = -1;
       index++;
     }
   }
   printf("finish the nt_create ...\n");
   //print the route table
   printf("\n\n********   The Neighbor table  ********\n");
   for(i=0; i<NbrCount; i++){
     printf("nodeID: %d -- nodeIP: %lu -- conn:%d\n", head[i].nodeID, (unsigned long)head[i].nodeIP, head[i].conn);
   }
   printf("**************  END  *************\n\n");
   return head;
}
```

(2) 删除路由表

```c
//这个函数删除一个邻居表. 它关闭所有连接, 释放所有动态分配的内存.
void nt_destroy(nbr_entry_t* nt){
  int NbrCount = topology_getNbrNum();
  int i;
  for(i=0; i<NbrCount; i++){
    close(nt[i].conn);
  }
  free(nt);
}
```

(3) 为指定的邻居分配 TCP 连接

```c
//这个函数为邻居表中指定的邻居节点条目分配一个TCP连接. 如果分配成功, 返回1, 否则返回-1.
int nt_addconn(nbr_entry_t* nt, int nodeID, int conn)
{
  int NbrCount = topology_getNbrNum();
  int i;
  for(i=0; i<NbrCount; i++){
    if(nt[i].nodeID==nodeID){
      nt[i].conn = conn;
      return 1;
      }
  }
  return -1;
}
```

-1.2- 对重叠网络的拓扑信息进行解析的文件

(1)返回主机 ID

```c
int topology_getMyNodeID()
{
  char hname[HOSTNAMESIZE];
  char *temp;
  char *buf;
  char *delim = ".";
  struct hostent *hent;
  gethostname(hname, sizeof(hname));
  hent = gethostbyname(hname);
  //inet_ntoa(*((struct in_addr *)h->h_addr_list[i]))
  buf = inet_ntoa(*((struct in_addr *)hent->h_addr_list[0]));
  temp = strtok(buf, delim);
  temp = strtok(NULL, delim);
  temp = strtok(NULL, delim);
  temp = strtok(NULL, delim);
  if(temp==NULL)    return -1;
  int ID = atoi(temp);
  return ID;
}
```

(2)根据主机名返回主机 ID

```c
int topology_getNodeIDfromname(char* hostname)
{
  char *buf;
  struct hostent *hent;
  hent = gethostbyname(hostname);
  //temp is ip string
  buf = (char *)inet_ntoa(*((struct in_addr *)hent->h_addr_list[0]));
  char *temp;
  char *delim = ".";
  temp = strtok(buf, delim);
  temp = strtok(NULL, delim);
  temp = strtok(NULL, delim);
  temp = strtok(NULL, delim);
  if(temp==NULL)  return -1;
  int ID = atoi(temp);
  //debug
  //printf("the ID is %d\n", ID);
  return ID;
}
```

(4) 根据 IP 地址返回主机的 ID

```c
int topology_getNodeIDfromip(struct in_addr* addr)
{
  char *buf;
  char *temp;
  char *delim = ".";
  //use inet_ntoa()
  buf = (char *)inet_ntoa(*addr);
  temp = strtok(buf, delim);
  temp = strtok(NULL, delim);
  temp = strtok(NULL, delim);
  temp = strtok(NULL, delim);
  if(temp==NULL)  return -1;
  int ID = atoi(temp);
  return ID;
}
```

其他函数具体实现请参考/topology/topology.c

-1.3-

需要实现的 SON 收发报文的 API 函数

所有节点等待 ID 比自己大的主机进入连接：

```
void* waitNbrs(void* arg) {
  //que ding ID bi zi ji da de lin ju de ge shu
  socklen_t clilen;
  int listenfd, connfd;
  struct sockaddr_in cliaddr, servaddr;
  if((listenfd = socket(AF_INET, SOCK_STREAM, 0)) <0 ){
    perror("Problem in creating the server socket...\n");
    exit(2);
  }
  servaddr.sin_family = AF_INET;
  servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
  servaddr.sin_port = htons(CONNECTION_PORT);
  bind(listenfd, (struct sockaddr *)&servaddr, sizeof(servaddr));
  listen(listenfd, 20);
  printf("SON level : Server running ... wait for connections ... \n");
  int i;
  int count = topology_getNbrNum();
  int hostID = topology_getMyNodeID();
  for(i=0; i<count; i++){
    if(nt[i].nodeID > hostID){
      clilen = sizeof(cliaddr);
      connfd = accept(listenfd, (struct sockaddr *)&cliaddr, &clilen);
      nt[i].conn = connfd;
      printf("SON level: Receive the request...%d\n", connfd);
    }
  }
  pthread_exit(NULL);
  return NULL;
}
```

所有站点请求连接到 ID 比自己小的所有的邻居

```
int connectNbrs() {
  int count = topology_getNbrNum();
  int hostID = topology_getMyNodeID();
  int i;

  for(i=0; i<count; i++){
    if(nt[i].nodeID < hostID){
      int sockfd;
      struct sockaddr_in servaddr;
      if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
            perror("Problem in creating the socket ...\n");
            exit(2);
      }
      memset(&servaddr, 0, sizeof(servaddr));
      servaddr.sin_family = AF_INET;
      servaddr.sin_addr.s_addr = nt[i].nodeIP;
      servaddr.sin_port = htons(CONNECTION_PORT);

      if(connect(sockfd,  (struct sockaddr*)&servaddr, sizeof(servaddr))< 0){
              perror("Problem in connecting to the server ...\n");
              exit(3);
      }
      nt[i].conn = sockfd;
     }
  }
  return 1;
}
```

每个站点都维护一个线程来持续的接收来自邻居的报文，并将接收到的报文发送给 SIP 进程，所有的监听邻居的进程都是在所有邻居的 TCP 连接全部建立好之后启动的：

```
void* listen_to_neighbor(void* arg) {
  //你需要编写这里的代码.
  printf("creat a listening thread ...\n");
  int index = *((int *)arg);
  printf("SON: listen to neighbor ID: %d CONN: %d\n", nt[index].nodeID, nt[index].conn);
  while(1){
    sip_pkt_t pkt;
    recvpkt(&pkt, nt[index].conn);
    forwardpktToSIP(&pkt, sip_conn);
  }
}
```

每个站点打开 TCP 端口 SON_PORT, 等待来自

```
void waitSIP() {
  socklen_t clilen;
  int listenfd = 0;
  struct sockaddr_in cliaddr, servaddr;

  listenfd = socket(AF_INET, SOCK_STREAM, 0);
  servaddr.sin_family = AF_INET;
  servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
  servaddr.sin_port = htons(SON_PORT);

  bind(listenfd, (struct sockaddr *)&servaddr, sizeof(servaddr));
  listen(listenfd, 20);
  printf("SIP level : Server running ... wait for connections ... \n");

  clilen = sizeof(cliaddr);
  sip_conn = accept(listenfd, (struct sockaddr *)&cliaddr, &clilen);
  printf("SIP level : Received request ... \n");
  while(1){
    sip_pkt_t pkt;
    int nextNode;
    getpktToSend(&pkt, &nextNode, sip_conn);

    if(nextNode == BROADCAST_NODEID){
      int Count = topology_getNbrNum();
      int i;
      for(i=0; i<Count; i++){
        sendpkt(&pkt, nt[i].conn);
      }
    }
```

```
    else{
        /*
         *fix the bug !!!
         */
        for(i=0; i<Count; i++){
          if(nt[i].nodeID == nextNode) break;
        }
        sendpkt(&pkt, nt[i].conn);
    }
  }
}
```

停止重叠网络的函数

```
//这个函数停止重叠网络, 当接收到信号SIGINT时, 该函数被调用.
//它关闭所有的连接, 释放所有动态分配的内存.
void son_stop() {
  printf("son_stop...\n");
  int Count = topology_getNbrNum();
  int i;
  for(i=0; i<Count; i++){
    close(nt[i].conn);
  }
  free(nt);
  exit(-1);
}
```

-1.4-

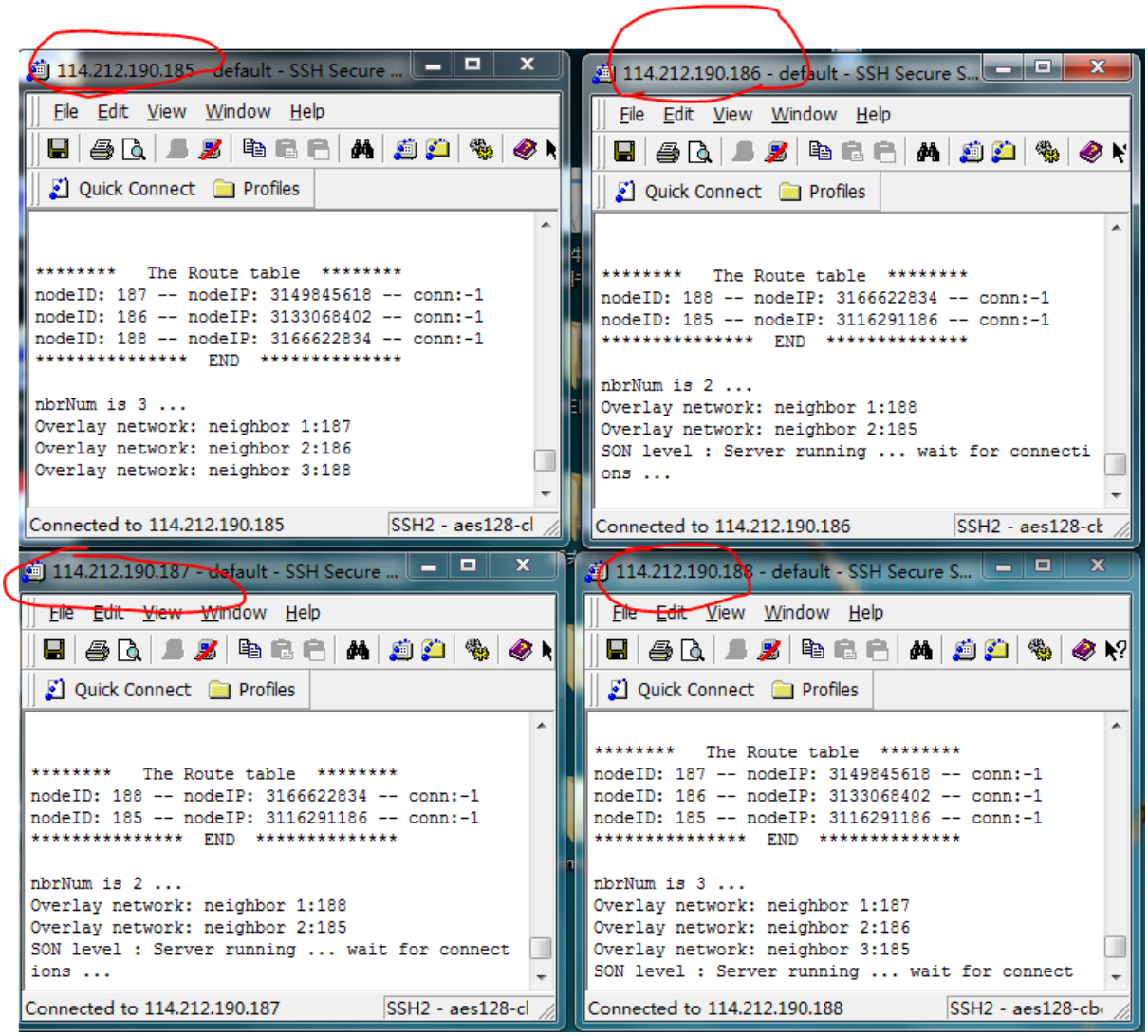
需要实现的 SIP 层函数

主要有 connectToSON()/ routeupdate_daemon()/ pkthandler()

具体实现请参见 /sip/sip.c

四、实验结果及报文分析

SON 层连接成功后打印的邻居表信息：



185 接收到的来自 186/187/188 的空的路由更新报文

186 接收来自 188 和 185 的路由更新报文

```
************** Route Update Info ******
****
Routing srcnode_ID: 187
Routing destnode_ID: 9999
Routing type : 1

************** Route Update Info ******
****
Routing srcnode_ID: 188
Routing destnode_ID: 9999
Routing type : 1

************** Route Update Info **********
Routing srcnode_ID: 186
Routing destnode_ID: 9999
Routing type : 1

************** Route Update Info **********
Routing srcnode_ID: 187
Routing destnode_ID: 9999
Routing type : 1

************** Route Update Info **********
Routing srcnode_ID: 188
Routing destnode_ID: 9999
Routing type : 1

************** Route Update Info **********
Routing srcnode_ID: 186
Routing destnode_ID: 9999
Routing type : 1
```

```
************** Route Update Info *********
*
Routing srcnode_ID: 188
Routing destnode_ID: 9999
Routing type : 1

************** Route Update Info **********
Routing srcnode_ID: 185
Routing destnode_ID: 9999
Routing type : 1

************** Route Update Info **********
Routing srcnode_ID: 188
Routing destnode_ID: 9999
Routing type : 1

************** Route Update Info **********
Routing srcnode_ID: 185
Routing destnode_ID: 9999
Routing type : 1
```

187 接收来自 185 和 188 的路由更新报文

188 接收来自 185/186/187 的路由更新报文

```
************** Route Update Info **********
Routing srcnode_ID: 185
Routing destnode_ID: 9999
Routing type : 1

************** Route Update Info **********
Routing srcnode_ID: 188
Routing destnode_ID: 9999
Routing type : 1

************** Route Update Info **********
Routing srcnode_ID: 185
Routing destnode_ID: 9999
Routing type : 1

************** Route Update Info **********
Routing srcnode_ID: 188
Routing destnode_ID: 9999
Routing type : 1
```

```
************** Route Update Info *******
***
Routing srcnode_ID: 186
Routing destnode_ID: 9999
Routing type : 1

************** Route Update Info *******
***
Routing srcnode_ID: 187
Routing destnode_ID: 9999
Routing type : 1

************** Route Update Info **********
Routing srcnode_ID: 185
Routing destnode_ID: 9999
Routing type : 1

************** Route Update Info **********
Routing srcnode_ID: 186
Routing destnode_ID: 9999
Routing type : 1
```

五、实验中遇到的问题

此处主要分析调试程序中遇到的一些问题：

BUG 1:

首先，connect()函数的返回值是 0 表示连接成功，accept（）函数的返回值 是服务器端接收连接后返回的套接字，在设置邻居表项的时候，我在客户端赋值时，赋值处理将 accept（）返回值赋值给 nt[i].conn，但是应该将 socket(AF_INT, SOCK_STREAM)的返回值赋值给 nt[i].conn，通过这次调试发现自己对于系统函数的理解太浅了，没有去深入理解调用这个函数的返回值是不同情况下分别表示什么意义，认识到以后学习知识不能不求甚解，否则学习成了蜻蜓点水，难成大事。

BUG2:

在写程序的时候发现自己还是一个不善于表达的人，现在越来越感觉，写程序就是表达自己，只有会表达自己，才可以用程序很好的表达自己的意思。

但是表达的好坏取决于自己对于自己想表达的目的以及自己是否对自己想表达的内容十分了解，如果自己都不清楚自己想表达什么，很难想象写出的程序的风格，肯定维护和调试起来花费巨大的精力。

BUG3:

Son 实验中主要实现的测试效果是打印每隔 5 秒钟发送的更新路由报文

但是最开始的时候由于在建立连接的问题和接受发送的套接字描述符有问题，会有奇怪的结果，又 2 个窗口是打印的特别快，因此我猜测如果发送的目的套接字描述符为 0 的话，应该会出现很奇怪的现象，具体情况需要参考网上的一些文档。

六、 实验的启示/意见和建议

通过此次实验我们实现了在根据拓扑文件，建立局域的拓扑连接，然后实现了在拓扑间按照一定的规则定时的发送路由更新报文，以检验我们实现的简单重叠网络 SON 能够正常的工作。

此次实验从开始做到调好花了 10 天吧！ 中间调试程序花了很久，而且还有编译原理实验!！伤不起啊！