计网课程设计报告

一．介绍：

我们组的课程设计按要求合理的完成了老师的要求，基于规定的拓扑图自行设计拓扑图，并且可以做到每台路由器基于 LS 和 DV 算法计算自己到其它路由器的最短耗费以及路径，某个路由器被 down 掉后其他的路由也会被通知到。之后的中心化的实施也没有问题。

二．设计思路：

我们的程序框架大致为，LS 算法在 RouteTableLS.cpp 中实现，DV 算法在 RouteTableDV.cpp 中实现。DataStructure.cpp 中存储数据结构，包括基本的地址信息，路由表信息单位，网络拓扑图中的路径信息（LS 算法用到）和路由器信息。VirtualPacket.cpp 用于将各种要发送出去的不同类型的数据打包成字符串的形式，同时具有将接受到的字符串（数据包）解析的功能。Controller.cpp 用于实现绑定套接字，监听对应的端口，各种包的接收，转发和以及确定邻居是否还存活的功能。最后 centerlizedController.cpp 和 distributeController.cpp 负责中心化功能的实现。

三．实施想法：

实现功能一定是要多线程的，我们的程序中一个线程用来监听信息，另一个线程用来发送普通的数据包，还有一个线程用于发送心跳检测包，用于测试其它路由器是否还活着。

```
int main() {

    iniHearbeatTimeTable();

    pthread_t tids[5];

    pthread_create(&tids[0], NULL, start, NULL);


    // 每隔5s发一次普通包
    pthread_create(&tids[1], NULL, send, NULL);

    pthread_create(&tids[2], NULL, heartBeat, NULL);

    pthread_exit(NULL);

    return 0;

}
```

这三个函数对应着三个线程函数的起始地址：

```
void *start(void *args) {
    c.listen();
    c.run();
}

void *send(void *args) {
    while (1) {
        Sleep(5000);
        srand((unsigned)time(NULL));

        int n = rand() % 5;
        while (n == localname - 'A') {
            n = rand() % 5;
        }
        c.sendNormalPacket(c.table.hostAddrs[n].ipaddress, "TEST PACKET");
    }
}

void *heartBeat(void *args) {
    int n = 0;
    while(1) {
        n++;
        Sleep(5000);
        c.sendHeartBeatPacket();
        if (n > 5)
            c.checkNeighbor();
```

关于 LS 算法：

LS 算法是预先知道拓扑图的，根据拓扑图提前算出每个路由器到其他路由器的最短路径，在进行 dijkstra 算法的时候要注意用一个 pre 数组来存储下一跳的路由，便于发送数据包的时候根据目的地去找到下一跳要发送信息的路由是哪个。

```
while (!q.empty()) {
    char temp = q.front();
    q.pop();
    for (int i = 0; i < networkGraph[temp].size(); i++) {
        char add2name = networkGraph[temp][i].addr2.name;
        if (!isVisit[add2name - 'A'- index_sub(have_delete, add2name)]) q.push(add2name);
        if (networkGraph[temp][i].cost + dis[temp - 'A'- index_sub(have_delete, temp)] < dis[add2name - 'A'- index_sub(have_delete, add2name)])
            dis[add2name - 'A'- index_sub(have_delete, add2name)] = networkGraph[temp][i].cost + dis[temp - 'A' - index_sub(have_delete, temp)]
            pre[add2name - 'A'- index_sub(have_delete, add2name)] = temp - 'A' - index_sub(have_delete, temp);
        }
    }
    isVisit[temp - 'A' - index_sub(have_delete, temp)] = true;
```

关于 DV 算法：

DV 算法根据距离向量来计算，每次都从邻居获取路由选择表，从而更新自己的路由选择表，这里的到达目的路由先要到达的下一跳是存储在路由选择表中的。

```
bool DValgorithm(vector<int> Nei_dis, char anHostName) {// 更新路由表，传入的是某个邻居发来的路由表以及
    vector<vector<int> > origin_table;
    copy(origin_table, myTable);
    //将邻居新信息来的距离向量更新自己的路由选择表
    for (int i = 0; i < hostAddrs.size(); i++) {
        myTable[anHostName - 'A'][i] = Nei_dis[i];
    }
    for (int i = 0; i < hostAddrs.size(); i++) {//Bellman-Ford方程
        for (int j = 0; j < hostAddrs.size(); j++) {
            for (int z = 0; z < hostAddrs.size(); z++) {
                if (myTable[i][z] + myTable[z][j] < myTable[i][j]) {
                    myTable[i][j] = myTable[i][z] + myTable[z][j];
                    //若本机的距离向量改变，则更新路由转发表
                    if (i + 'A' == myHostName && myTable[i][j] < 9999) {
                        //若路由转发表为空，直接加入
                        if (routetable.empty()) {
                            Addr destAddr = Addr(hostAddrs[j]);
                            char nextRoute[16];
                            strcpy(nextRoute, hostAddrs[z].ipaddress);
                            routetable.push_ba
                                vector<Addr> RouteTableDV::hostAddrs
                                vector<Addr> hostAddrs
                                e, myTable[i][j]));
                        }
                        else {
                            bool is = false;
                            //若路由转发表中已经存在目的地，则直接修改路由转发表中的信息
                            for (int k = 0; k < routetable.size(); k++) {
                                if (routetable[k].addr.name == hostAddrs[j].name) {
                                    bool is1 = false;

    else {
        bool is = false;
        //若路由转发表中已经存在目的地，则直接修改路由转发表中的信息
        for (int k = 0; k < routetable.size(); k++) {
            if (routetable[k].addr.name == hostAddrs[j].name) {
                bool is1 = false;
                //检测路由转发表中是否存在到中间点z的转发项，若有则更新转发表，将i到j的下一跳路由器设
                for (int m = 0; m < routetable.size(); m++) {
                    if (routetable[m].addr.name == hostAddrs[z].name) {
                        strcpy(routetable[k].nexthop, routetable[m].nexthop);
                        routetable[k].cost = myTable[i][j];
                        is1 = true;
                        break;
                    }
                }
                //不存在到中间点z的转发项，则i到j的下一跳就为z
                if (!is1) {
                    strcpy(routetable[k].nexthop, hostAddrs[z].ipaddress);
                    routetable[k].cost = myTable[i][j];
                }
                is = true;
                break;
            }
        }

    if (!is) {
        //寻找本机到中间路由z的下一跳
        for (int m = 0; m < routetable.size(); m++) {
            //将i到j的下一跳变为i到z的下一跳
            if (routetable[m].addr.name == hostAddrs[z].name) {
                Addr destAddr = Addr(hostAddrs[j]);
                char nextRoute[16];
                strcpy(nextRoute, routetable[m].nexthop);
                routetable.push_back(routeTableEntry(destAddr, nextRoute, myTable[i][j]));
            }
        }
    }
}
```

当某台路由器被 down 掉时，路由器与路由器之间都会发送心跳检测包，最终某台路由器被 down 掉的信息会被其他所有路由器知道，在 LS 算法中其他路由器会直接删除对应的边，形成新的拓扑图，然后重新调用 LS 算法，计算到其它路由的最短路径。

DV 算法则是将与被 down 掉的路由器直接相连路径的耗费都设为 9999，并且与被 down 路由器相连的路由器会通知其邻居那台被 down 掉的路由器无法到达，以防止毒性逆转的情况发生。

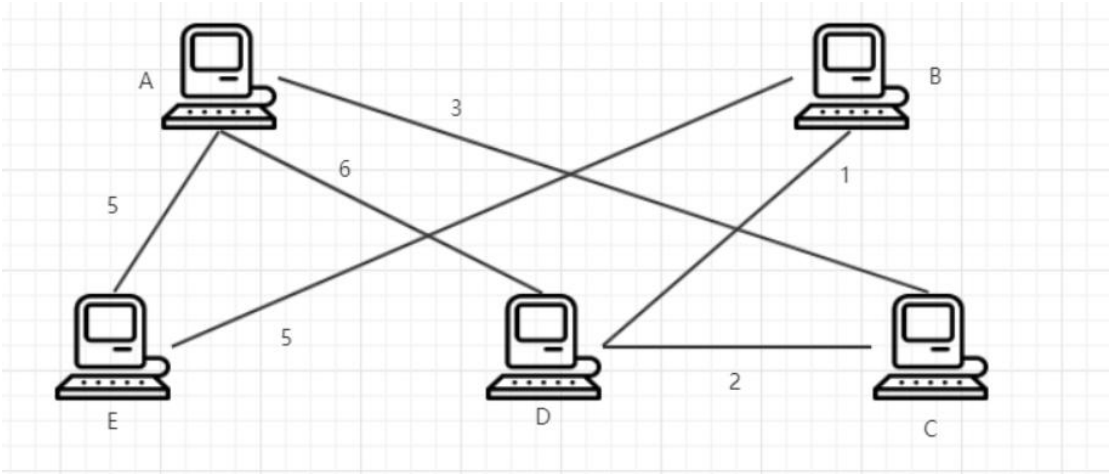
关于心跳检测的思路：

这一部分我们组的成员都有不同的想法，我们最终选择了一种比较切合实际的想法，就是每台路由器给其他路由器发送心跳包，同时也接收每台路由器发送过来的心跳包并记录时间，如果当前的时间减去某个路由器最后一次发送数据包的时间大于两秒，就说明对应的路由被 down 掉了。

```
void checkNeighbor() {
    time_t currentTime;
    time(&currentTime);
    bool isChange = false;
    for (auto p : heartBeatTimetable) {
        double timeDiff = difftime(currentTime, p.second);
        if (timeDiff > 2.0) {
            cout << "Router is down: " << table.getHostName(p.first.ipaddress) << endl;
            if (table.setdown(table.getHostName(p.first.ipaddress))) {
                isChange = true;
                //sendDownPacket(p.first);
                table.print();
            }
        }
    }
    // DV
    if (isChange) {
        //sendUpdatePacket();
    }
}
```

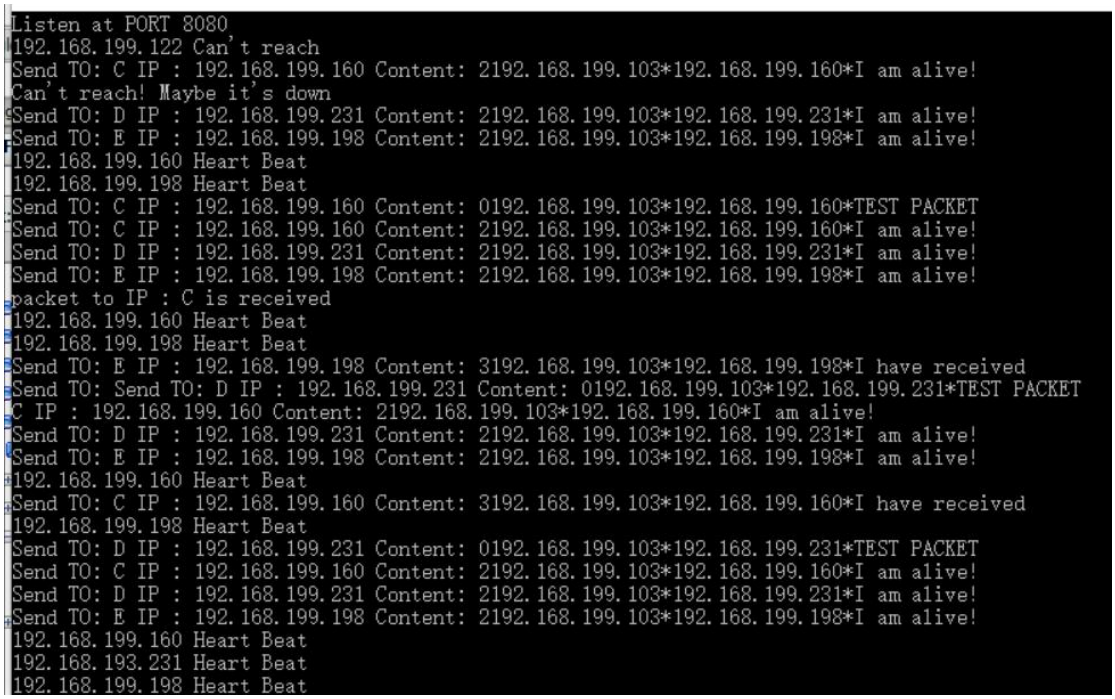
四．测试结果：

测试思路：我们的测试思路是先保持最初的拓扑图：（耗费是我们自己规定的）

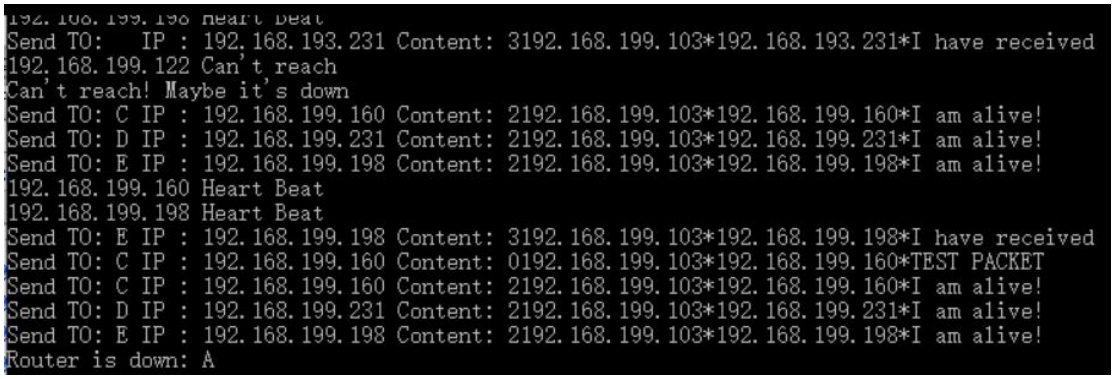


然后过一阵子 down 掉 B，之后再 down 掉 C，整个过程中都在查看各个路由器报文以及路由表情况。

关于路由器 A 的效果图：（DV 算法）
可以看出路由器 A 给其他路由器正常的发送报文，以及自己可以接收到报文。



当路由器 B 被 down 掉时，会显示不可达，并且会有报文发送到 A 通知路由器 B 被 down 掉了。



路由器 A 的 LS 算法效果图：

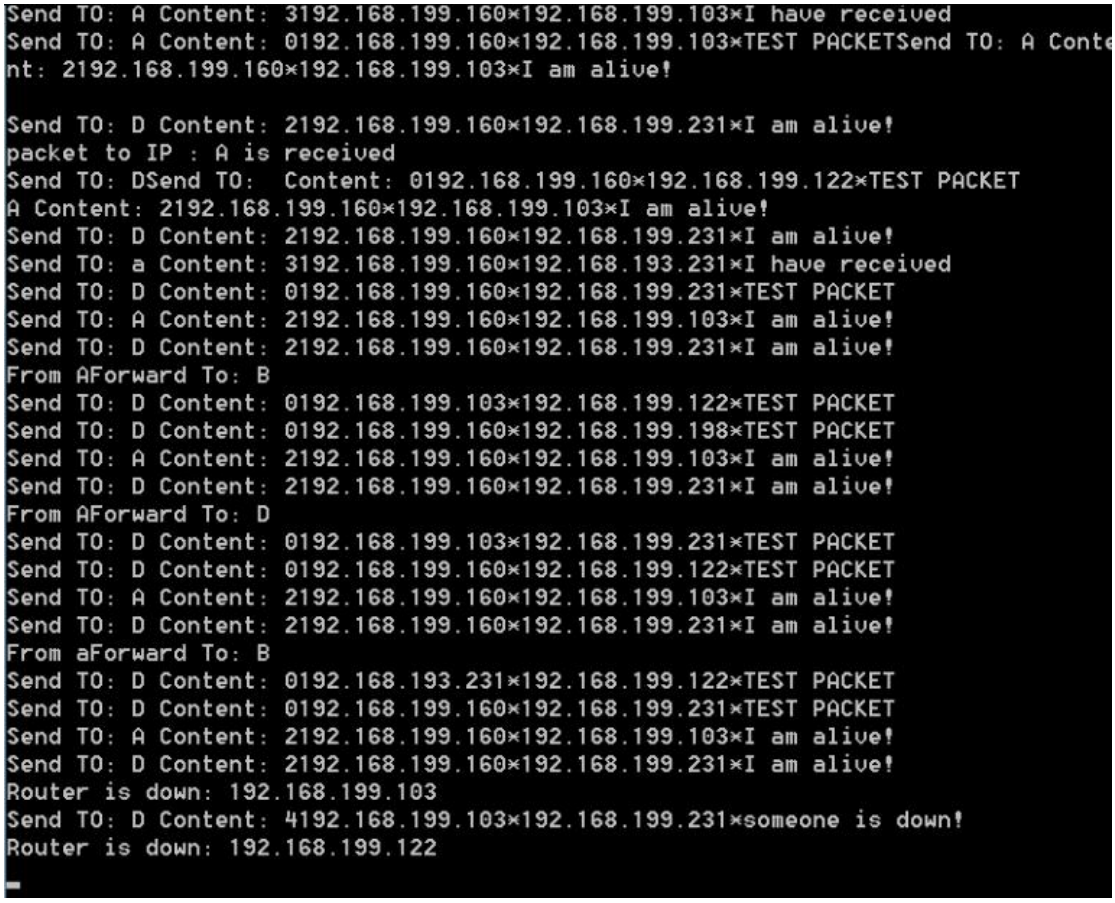
```
Router is down: B
Send TO: C Content: 4192.168.199.122*192.168.199.160*someone is down!
Send TO: D Content: 4192.168.199.122*192.168.199.231*someone is down!
Send TO: E Content: 4192.168.199.122*192.168.199.198*someone is down!
-------------------------
destName: A
NExtIP: 192.168.199.103
Cost: 0
destName: C
NExtIP: 192.168.199.160
Cost: 3
destName: D
NExtIP: 192.168.199.160
Cost: 5
destName: E
NExtIP: 192.168.199.198
```

当 B 被 down 掉时，并且 A 的路由选择表中只会有 A 到 C，D，E 的最少消耗以及到达目的地前（或者是直接到达目的地）下一跳路由的 IP 地址。A 的路由选择表中关于 B 的信息也都会被删除掉。

```
-------------------------
destName: A
NExtIP: 192.168.199.103
Cost: 0
destName: C
NExtIP: 192.168.199.160
Cost: 3
destName: D
NExtIP: 192.168.199.160
Cost: 5
destName: E
NExtIP: 192.168.199.198
Cost: 5
-------------------------
Router is down: C
Send TO: D Content: 4192.168.199.160*192.168.199.231*someone is down
Send TO: E Content: 4192.168.199.160*192.168.199.198*someone is down
-------------------------
destName: A
NExtIP: 192.168.199.103
Cost: 0
destName: D
NExtIP: 192.168.199.231
Cost: 7
destName: E
NExtIP: 192.168.199.198
Cost: 5
-------------------------
```

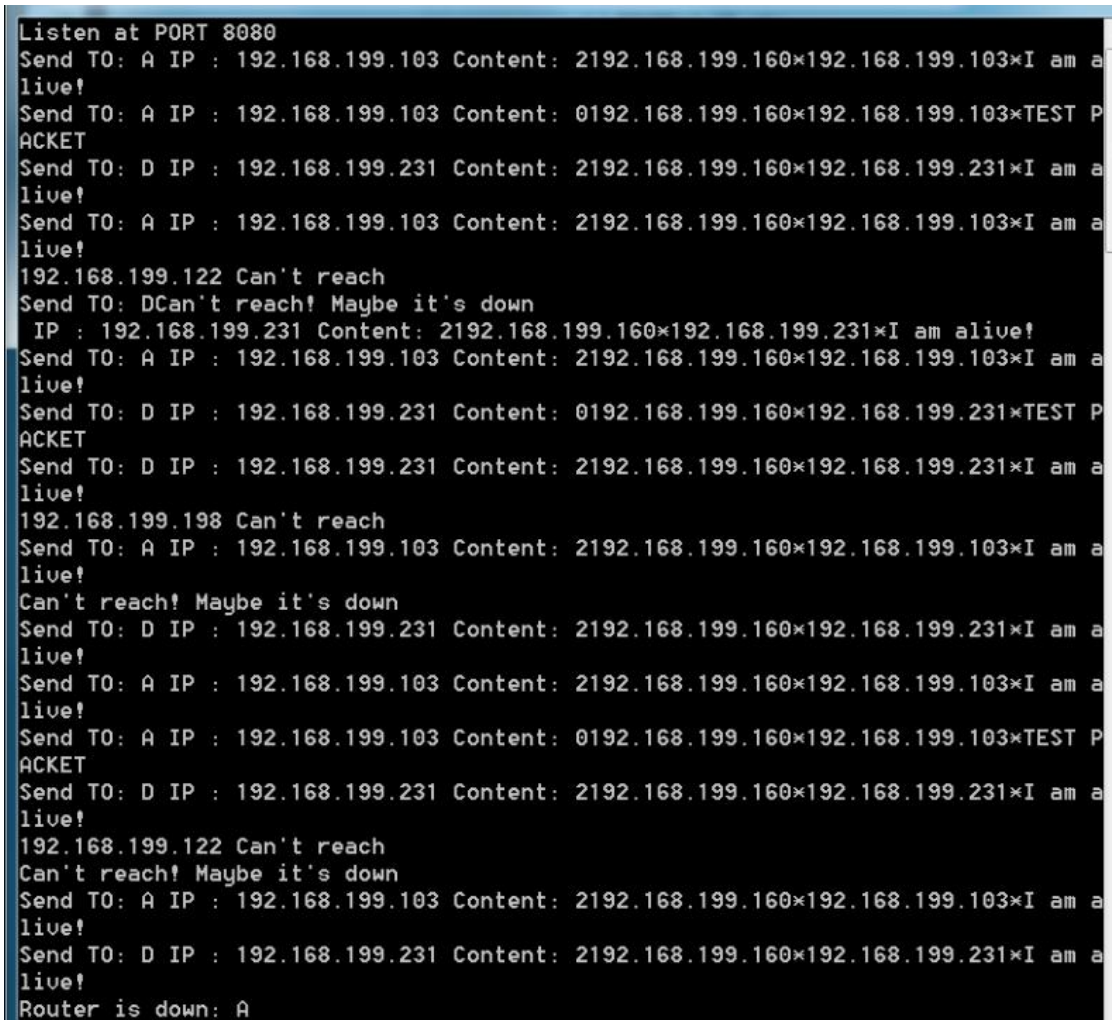
当 C 又被 down 掉之后，A 收到这个消息后它的路由选择表会更新，关于 C 的都会被删除。

关于路由器 C 的效果图：（LS 算法）



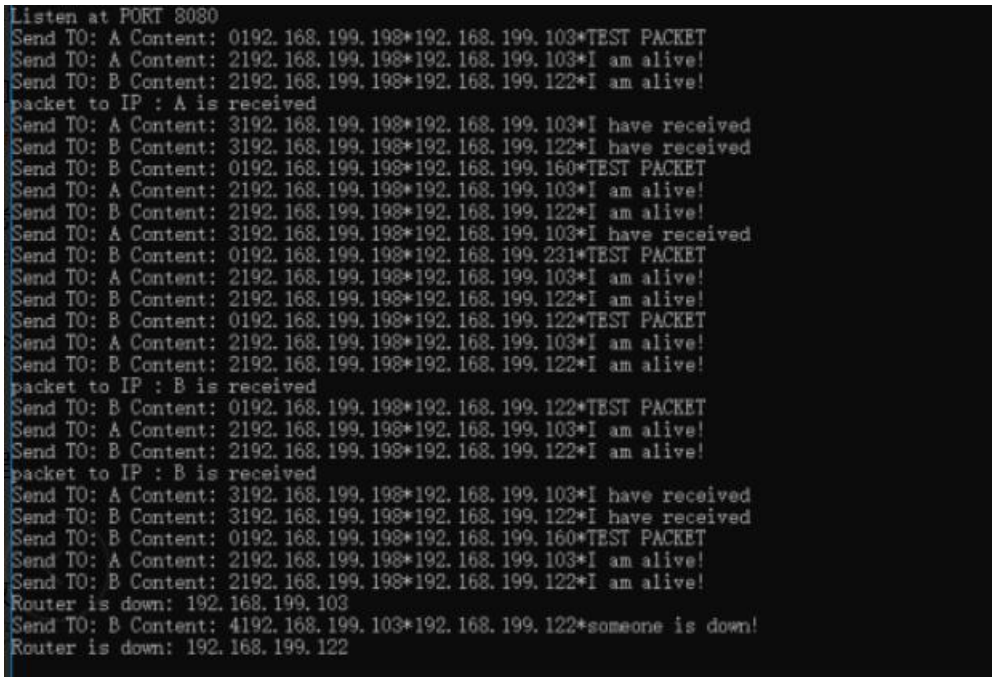
B 被 down 掉了之后路由器 C 会接受到 B（ip 为 192.168.199.122）被 down 掉的报文。之后我们手动 down 掉了 C（关闭程序），所以运行截图只有这些。

关于路由器 C 的效果图（DV 算法）



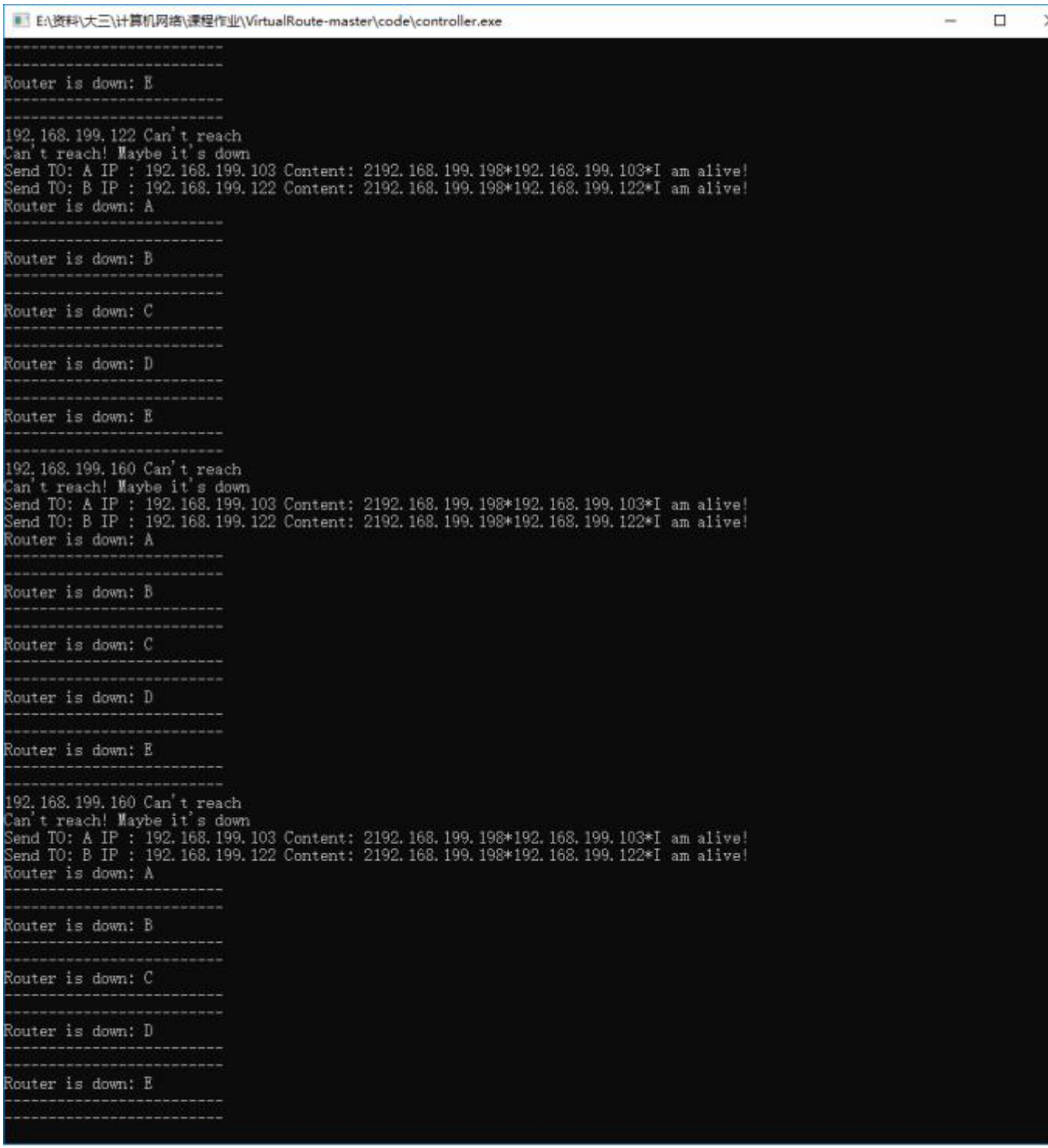
可以看到路由器 B 被 down 掉之前 C 可以发送接收并发送报文给任意路由器，之后会受到 B 被 down 掉的消息。

路由器 E 的效果图（LS 算法）



路由器 E 的 DV 算法效果图：

这里的测试是当其它的主机全部 down 掉时路由器 E 的路由表状况以及向其他路由再发送报文的状况（都为不可达）。


E:\资料\大三\计算机网络\课程作业\VirtualRoute-master\code\controller.exe
Router is down: E
192.168.199.122 Can't reach
Can't reach! Maybe it's down
Send TO: A IP : 192.168.199.103 Content: 2192.168.199.198*192.168.199.103*I am alive!
Send TO: B IP : 192.168.199.122 Content: 2192.168.199.198*192.168.199.122*I am alive!
Router is down: A
Router is down: B
Router is down: C
Router is down: D
Router is down: E
192.168.199.160 Can't reach
Can't reach! Maybe it's down
Send TO: A IP : 192.168.199.103 Content: 2192.168.199.198*192.168.199.103*I am alive!
Send TO: B IP : 192.168.199.122 Content: 2192.168.199.198*192.168.199.122*I am alive!
Router is down: A
Router is down: B
Router is down: C
Router is down: D
Router is down: E
192.168.199.160 Can't reach
Can't reach! Maybe it's down
Send TO: A IP : 192.168.199.103 Content: 2192.168.199.198*192.168.199.103*I am alive!
Send TO: B IP : 192.168.199.122 Content: 2192.168.199.198*192.168.199.122*I am alive!
Router is down: A
Router is down: B
Router is down: C
Router is down: D
Router is down: E