天津大学

《计算机网络》课程设计报告

小组作业：路由器算法实验

学　号 6319000359

姓　名 张明君

学　号 6319000411

姓　名 唐琳

学　院 智能与计算学部

专　业 计算机科学与技术

年　级 2019 级

任课教师 赵增华

2021 年 12 月 7 日

DV 算法的原理和过程

DV 算法是分布式的基于距离向量的路由选择协议。网络中的每一个路由器都维护着从他自己到其他每一个目的网络的距离记录（这是一组距离，称为距离向量）。其中距离也成为条数或者是代价在一开始的 时候，一个路由器到自己网络的条数是 1。每经过一个路由器，距离就加一。

具体的过程如下：

1. 对地址为 x 的相邻路由器发来的 RIP 报文，修改此报文中的所有项目：把“下一跳”字段中的地址改为 X，并把所有的“距离”字段+1。
2. 对修改后的 RIP 报文中的每一个项目，进行以下步骤：
   （1）R1 路由表中若没有 Net3，则把该项目填入 R1 路由表
   （2）R1 路由表中若有 Net3，则查看下一跳路由器地址： 若下一跳是 x，则用收到的项目替换源路由表中的项目；若下一跳不是 x，原来距离比从 x 走的距离 远则更新，否则不作处理。
3. 若 180s 还没收到相邻路由器 X 的更新路由表，则把 X 记为不可达的路由器，即把距离设置为 16。
4. 返回

算法实现规划

在本实验中算法的实验更具实验指导书进行，首先完成了每个节点的初始化函数和更新路由表函数。再将这些函数复制到不同的节点中，来实现其他节点的初始化和更新路由表函数。

实现具体细节

这里以实现 `node0` 为例，介绍实现细节。

`rtinit0`

```
void rtinit0() {
    // 根据拓扑结构图进行初始化操作
    linkCost0[0] = 0;
    linkCost0[1] = 1;
    linkCost0[2] = 3;
    linkCost0[3] = 7;

    memcpy(dt0.costs[NODE0], linkCost0, sizeof(int[4]));
    memcpy(minCost0, linkCost0, sizeof(int[4]));
    sendDV0 = FALSE;

    printdt0();
    sendToNeighbors0(NODE0, minCost0);  // 发送给相邻的路由器
}
```

在一开始初始化的时候，将和该路由器直连的网络可以直接写在路由表上。再将 `linkCost0` 中的数据复制到路由表 0 和最小的 cost 上。打印完毕之后，将当前的路由表广播给相邻的路由器。

`update0`

```
void rtupdate0(struct rtpkt *rcvdpkt) {
    //对路由表更新
    if (isNeighbor0(rcvdpkt->sourceid)) {
        int i;
        for (i = 0; i < 4; i++) {
            if (minCost0[i] > rcvdpkt->mincost[i]) {
                int possibleRoute =
                    minCost0[rcvdpkt->sourceid] + rcvdpkt->mincost[i];
                if (possibleRoute < minCost0[i]) {
                    minCost0[i] = possibleRoute;
                    sendDV0 = TRUE;
                }
            }
        }
        memcpy(dt0.costs[NODE0], minCost0, sizeof(int[4]));
        memcpy(dt0.costs[rcvdpkt->sourceid], rcvdpkt->mincost, sizeof(int[4]));
        printdt0();
    }
    if (sendDV0) {
        sendToNeighbors0(NODE0, minCost0);
        sendDV0 = FALSE;
    }
}
```

对于路由表的更新函数的思路也很简单，首先，如果传入的路由表的相应的跳数小于该路由器中本身的条数，在去计算更新以后的跳数能否比当前的条数还要小，如果是的话，就更新路由表。这里使用嵌套的 `if` 语句来实现。如果更新了路由表，把 `sendDV0` 这个标志位记为 `true`，之后再去和其他路由器广播。

实现过程中遇到的问题

在更新路由表的函数中，一开始没有 `sendDV0` 这个变量，这样每一次即使没有更新路由表，也会将自己的路由表广播出去，形成的死循环。解决办法就是设置 `sendDV0` 这个标识位，只有当前路由器的路由表更改了以后才会将自己的路由表再次广播出去。

网络仿真的过程和结果

首先，书写了 `Makefile` 文件，键入 `mingw32-make` 进行编译，之后键入 `./all` 来执行编译之后 的文件。

执行结果：

```
PS G:\AlgorithmExercises\exercises> ./all
Enter TRACE:1
                via
   D0 |    0      1     2     3
  ----|-----------------------
     0|    0      1     3     7
     1|    0      0     0     0
dest 2|    0      0     0     0
     3|    0      0     0     0
At time t=0.000, node 0 sends packet to node 1 with: (0  1  3  7)
At time t=0.000, node 0 sends packet to node 2 with: (0  1  3  7)
At time t=0.000, node 0 sends packet to node 3 with: (0  1  3  7)
                via
   D0 |    0      1     2     3
  ----|-----------------------
     0|    0      1     2     7
     1|    1      0     1    16
dest 2|    0      0     0     0
     3|    0      0     0     0
At time t=0.998, node 0 sends packet to node 1 with: (0  1  2  7)
At time t=0.998, node 0 sends packet to node 2 with: (0  1  2  7)
At time t=0.998, node 0 sends packet to node 3 with: (0  1  2  7)
node2 distance_table: 3 1 0 2
                via
   D0 |    0      1     2     3
  ----|-----------------------
     0|    0      1     2     4
     1|    1      0     1    16
dest 2|    3      1     0     2
     3|    0      0     0     0
At time t=1.685, node 0 sends packet to node 1 with: (0  1  2  4)
At time t=1.685, node 0 sends packet to node 2 with: (0  1  2  4)
At time t=1.685, node 0 sends packet to node 3 with: (0  1  2  4)
```

在一开始的时候，路由器 0 的路由表只有和自己直连的路由器和到达的跳数，这是在路由器初始化的时候就执行完毕的。之后再将相邻的路由表进行相互传播，可以看到再截图中，路由器 0 和其他三个路由器转发了自己的路由表之后，路由表的变化。反复这个过程，之后可以得到一张不在变化的路由表。在路由 表中，16 表示不可达，所以在初始化的时候，也将不可达的路由器之间的跳数记为 16。

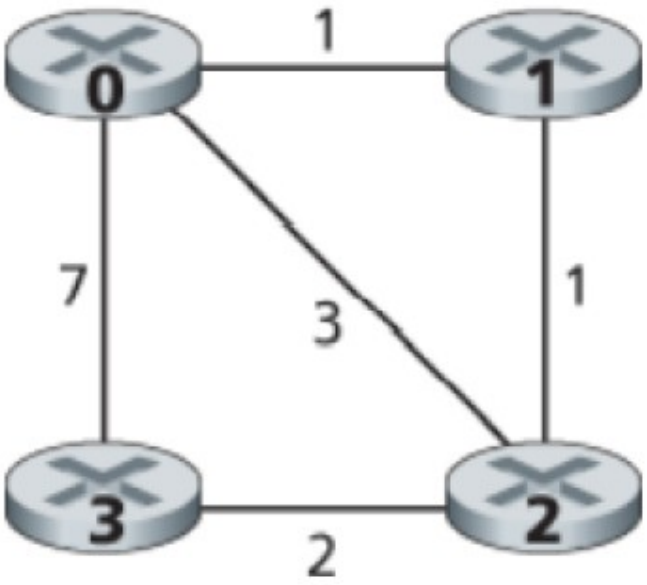
最后结果：

```
                via
   D0 |    0      1     2     3
  ----|-----------------------
     0|    0      1     2     4
     1|    1      0     1     3
dest 2|    2      1     0     2
     3|    5      3     2     0
node2 distance_table: 2 1 0 2
                via
   D0 |    0      1     2     3
  ----|-----------------------
     0|    0      1     2     4
     1|    1      0     1     3
dest 2|    2      1     0     2
     3|    4      3     2     0
node1 distance_table: 1 0 1 3

Simulator terminated at t=20003.199219, no packets in medium
```

算法功能测试

算法功能测试的结果可以在上一节中看到。 对于下图的网络拓扑结构，通过计算和上一节中的结果是一致的，所以可以得出算法的功能是完备的。



对结果的分析和总结

通过 DV 算法的实验，加深了我对于 DV 算法的理解，学会了 DV 算法的具体实现。

该算法实现并不困难，重点是要看懂算法的执行过程。还有就是要避免一些简单的错误，可能会导致重大的 bug，需要花费大量的时间去 debug，养成良好的代码习惯，可以提高编写代码的效率和效果。