Distance Vector Routing

My initial approach to this protocol was to write it as a simulation first and then attempt an actual implementation. As I worked through it, however, it became clear to me that the simulation would work identically to an actual implementation. I mention this because, although my project was designed and tested on one computer (I only have one computer suitable for testing), it should run well across multiple machines, provided they're all listening to the same port. There would obviously have to be work done to account for network topology changes and actual naming conventions (here, nodes are addressed by alphabetic characters instead of IP or MAC addresses, and cost calculation is somewhat arbitrary), and, instead of broadcasting its routing table every 15 seconds, each node would have to align its broadcast timing upon reception of routing information from other nodes, but the core of the actual protocol is ready to be expanded upon.

Each node running the algorithm maintains a Dictionary containing both its own routing information and the distance vectors of its neighbors. The structure used to contain this this information is:

$$RoutingTable\ = Dictionary < char, Dictionary < char, Node \gg$$

In this structure, the key of the outer Dictionary indicates the router from which we are traveling, and the key of the inner Dictionary (the value of the outer Dictionary) indicates the router to which we are traveling. The value of the inner dictionary (Node) is a Class which maintains the next-hop-node in the path and cost to travel from a router (whose address is given by the outer Dictionary's key) to another router (whose address is given by the inner Dictionary's Key).

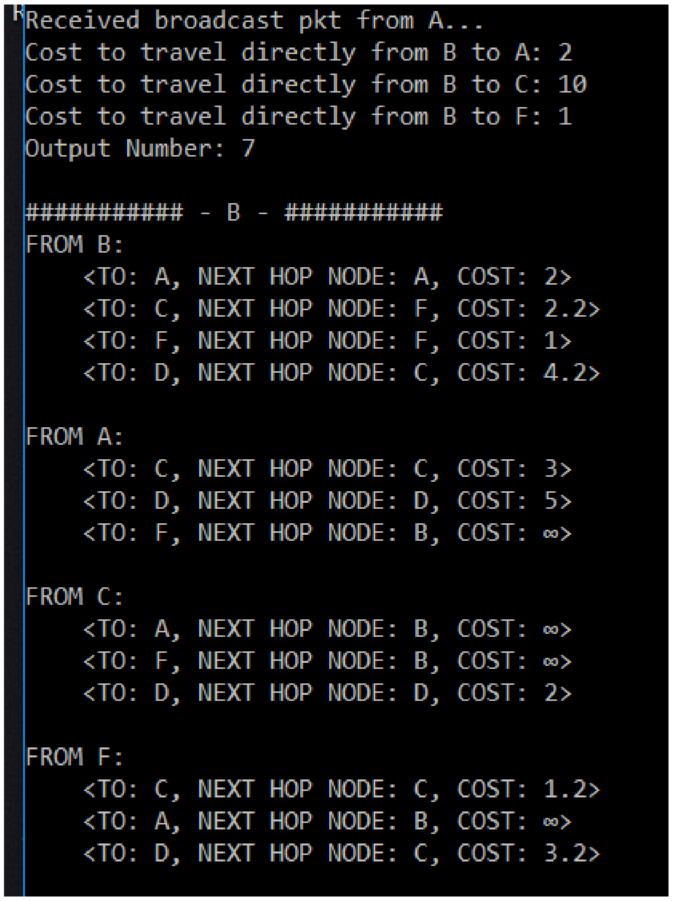
Initially, the first row, or the first entry in the $RoutingTable$ of any given node has a Key which is the name of the router (as indicated by the Input File) and a $Value$ which contains information on all other directly connected nodes (as indicated by the Input File). For example, an Input File named $A.txt$ contains:

```
3
B 2.3
C 5.34
D 3.0
```

If an instance of this project receives this Input File, its $RoutingTable$ will contain a $Key\ =\ A$ which indexes a $Value$ Dictionary with $Keys\ of\ B, C, D$ whose values are Nodes with $Cost$

parameters of $2.3, 5.34, and\ 3.0$, respectively. The RoutingTable will also include entries with $Keys\ of\ B, C, D$ with null values which will eventually hold the distance vector routing information of those nodes when it receives a packet containing that information from those routers. As a side note, if this were the actual Distance Routing protocol, the initial count of nodes in the $RoutingTable$ could expand if a node received a packet from a new node, but in this protocol, the amount of directly connected routers to a given node is limited to the amount contained in the first line of initial Input File (in this case, 3).

The Node Class contains four values – three characters and one float. The three character values are $From, To, Jump$, where From is the node from which we route, To is the node to which we route, and $Jump$ is the next-hop-node to which we initially jump to eventually arrive at the destination node To. The float value is the total cost of the of the current route from a router to its destination, and it will always be set to the minimum possible cost, given what the router knows about the link costs of the network. Below is a sample output from a node B; I'll define its output to illustrate the effectiveness of the implementation:



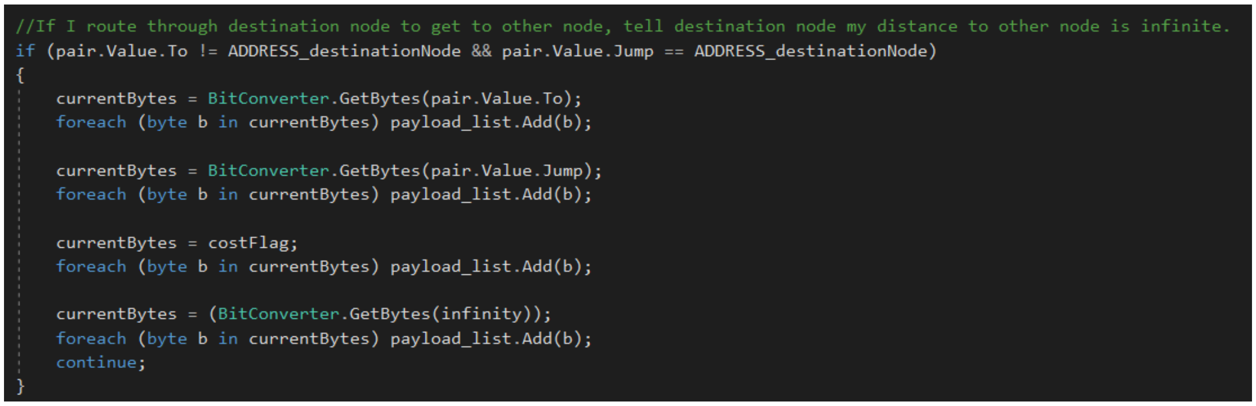
```
Received broadcast pkt from A...
Cost to travel directly from B to A: 2
Cost to travel directly from B to C: 10
Cost to travel directly from B to F: 1
Output Number: 7

########### - B - ###########
FROM B:
    <TO: A, NEXT HOP NODE: A, COST: 2>
    <TO: C, NEXT HOP NODE: F, COST: 2.2>
    <TO: F, NEXT HOP NODE: F, COST: 1>
    <TO: D, NEXT HOP NODE: C, COST: 4.2>

FROM A:
    <TO: C, NEXT HOP NODE: C, COST: 3>
    <TO: D, NEXT HOP NODE: D, COST: 5>
    <TO: F, NEXT HOP NODE: B, COST: ∞>

FROM C:
    <TO: A, NEXT HOP NODE: B, COST: ∞>
    <TO: F, NEXT HOP NODE: B, COST: ∞>
    <TO: D, NEXT HOP NODE: D, COST: 2>

FROM F:
    <TO: C, NEXT HOP NODE: C, COST: 1.2>
    <TO: A, NEXT HOP NODE: B, COST: ∞>
    <TO: D, NEXT HOP NODE: C, COST: 3.2>
```

This is the $7th$ output of this node, which means that it has received exactly 7 packets. This is the routing information which was updated immediately after receiving a packet from node A. Notice that for every neighboring node whose $NEXT\ HOP\ NODE$ is B, that neighbor tells B that the cost to travel to that node is infinite. This prevents the Poisoned Reverse problem, which will be discussed later.

Here we are interested in ensuring the correctness of $B's$ distance vectors, listed directly below $FROM\ B$: in the output, which are the vectors that will be broadcast to its neighbors $A, C, and\ F$. To travel from $B\ to\ A$, we see that the $NEXT\ HOP\ NODE$ is A, meaning that we travel directly from $B\ to\ A$, in one hop, and the cost is 2. It is easy to tell that this is the minimum cost because every other router that B has received information from routes to $A\ through\ B$ and tells B that their cost to route to A is $infinite$. To travel from $B\ to\ C$, the $NEXT\ HOP\ NODE$ is F and the cost is 2.2. This is also true, because routing directly to C from B has a cost of 10, and $F's$ cost to route to C is 1.2 while B can route to F for a cost of 1. I will not compare the minimum value achieved here to all other possible routes, but it can be proven that it is the minimum using the same logic.

Routing from $B\ to\ D$ may look a little strange, though, so I will explain its cost, which is correctly minimized to 4.2, in detail. For the sake of this example, let's initialize our cost variable $COST\ to$ 0. We see that our $NEXT\ HOP\ NODE$ to D is C, so we look up in our $RoutingTable$ how to get to C from B. Here, we find that our $NEXT\ HOPE\ NODE$ to C is F, so we look up F, and since the $NEXT\ HOP\ NODE$ to F is F, we add the cost to travel from B to C to $COST$. At this point, $COST$ is 2.2, and we've figuratively traveled to node C. At node C, the cost to travel directly to D is 2, so we add 2 to our $COST$ to get 4.2, which is the properly minimized value.

The Poisoned Reverse problem is tackled in the $ConstructPayload$ method of $DistanceRouter\_Simulation.cs$. Since each payload is constructed specifically for an intended destination node, we make a comparison between a node's To field and the address of the destination node and a node's $Jump$ field and the address of the destination node. If we are not routing to the node receiving the packet but we are jumping to the node receiving the packet, we set the cost of the distance vector to the floating point value for infinity:



```
//If I route through destination node to get to other node, tell destination node my distance to other node is infinite.
if (pair.Value.To != ADDRESS_destinationNode && pair.Value.Jump == ADDRESS_destinationNode)
{
    currentBytes = BitConverter.GetBytes(pair.Value.To);
    foreach (byte b in currentBytes) payload_list.Add(b);

    currentBytes = BitConverter.GetBytes(pair.Value.Jump);
    foreach (byte b in currentBytes) payload_list.Add(b);

    currentBytes = costFlag;
    foreach (byte b in currentBytes) payload_list.Add(b);

    currentBytes = (BitConverter.GetBytes(infinity));
    foreach (byte b in currentBytes) payload_list.Add(b);
    continue;
}
```

Detailed comments are included in the code that define every method and the more complex statements within the project (indexing a Dictionary within a Dictionary can be a little confusing), but if any more in-depth explanation is needed, I am available for that explanation. My implementation was derived from the power-point presentations uploaded to Canvas that describe this algorithm and from in-class notes. I did not reference specific material to write this code other than my previous two projects; however, those projects did reference and cite source material that does not belong to me. I doubt it's possible to make any good direct comparison between this project and the references of those projects, however, so it is not necessary to cite them again. Regardless, in-line citations will be included as comments if deemed necessary. Thank you for your time.