Project 3: Distance Vector Routing Algorithm

1 Introduction

In this project you will be implementing the **distance vector (DV) routing algorithm** as we discussed in class. Instead of implementing this algorithm directly in the operating system, you will use TCP or UDP to simulate the process. You can use **any programming language** that you are familiar with for this project.

2 Build Virtual Network

A network topology we used to discuss routing algorithms in class is shown on the right. To implement the routing algorithm over this network topology, we need to build a virtual network first.

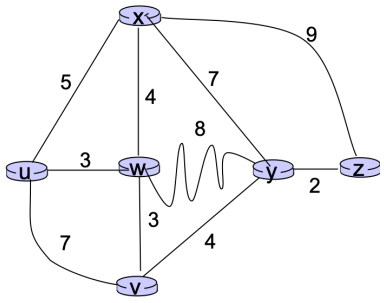
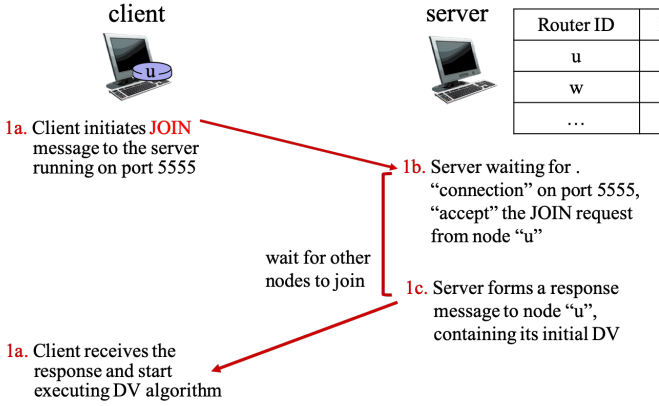


Figure 1: Network Topology

During the execution of DV algorithm, each node (device) needs to exchange their own distance vectors with their neighbors. In other words, it is required that each node has information of the nodes they directly connected with, including their IP addresses, port numbers etc. For simplification, we will use a **client-server communication paradigm** to simulate this process. The process is explained with the datagram below. Since all the nodes have the same behaviors, we use node "u" as an example for explanation.



Router ID	IP	port	List of <node, cost> pairs
u			<x, 5>, <w, 3>, <v,7>, <y, -1>, <z, -1>
w			<u, 3>, <x, 4>, <v,3>, <y, 8>, <z, -1>
…			…

Figure 2: C-S Interactions Before DV Execution

2.1 Routers Join Virtual Network

In Figure 2, node "u" runs the client process and it communicates with a server that runs on port 5555. (Note that this interaction happens after the client and server have established a connection if TCP is used.) The server maintains a table similar to the one shown in the figure, which contains topology information. (IP and port are initialized empty since they will be obtained from the client upon connection establishment). Client sends a `JOIN` message to the server to let the server know that the client is ready to execute DV algorithm. Upon the receipt of this message, server needs to do the following things:

- If UDP is used, extracts IP and port numbers from the message and fill them in the table maintained by the server; (Skipped if TCP is used; For TCP, this step is done when the server accepts the connection request from the client.)
- waits for other nodes to join before it sends back response.

In the response message, the server should send back the list of <node, cost> tuples that corresponds to the router ID (specified in the `JOIN` message). After the client receives the response, it could move on to a loop that runs the DV algorithm.

For table initialization, server should load **a configuration file** with the following format:

Router ID	List of <Node, cost> Pairs
u	<x, 5>, <w, 3>, <v, 7>, <y, -1>, <z, -1*>
x	<u, 5>, <w, 4>, <v, -1>, <y, 7>, <z, 9>
w	<u, 3>, <x, 4>, <v, 3>, <y, 8>, <z, -1>
v	<x, -1>, <w, 3>, <u, 7>, <y, 4>, <z, -1>
y	<x, 7>, <w, 8>, <u, -1>, <v, 4>, <z, 2>
z	<x, 9>, <w, -1>, <u, -1>, <y, 2>, <v, -1>

* -1 represents that the two nodes are not directly connected

Note that the list of tuple <node, cost> is not a distance vector and it does not need to be updated during the execution of the DV algorithms.

2.2 Routers Executes DV Algorithm

When the client receives the last response message from the server in Figure 2, client extract the list of tuples and use it to update its own distance vector. After that client is ready to execute the DV algorithm. Figure 3 shows the interactions between client and server during this time.

If the client has a updated DV, it sends a `UPDATE` message to the server. Upon the receipt of this message, server will forward this message to the neighbor nodes of the client, i.e., node "x", "w" and "v" in this example. Meanwhile, if server receives `UPDATE` message from node "x", "w" or "v", it needs to forward the message back to the client. Then client needs to update its own distance vectors again if needed and repeat this process. Note that during this process, the client also needs to maintain distance vectors for node "x", "w" and "v" as discussed in class. Eventually when the algorithm stabilizes, the client could exit the loop and terminate the connection with the server if needed.

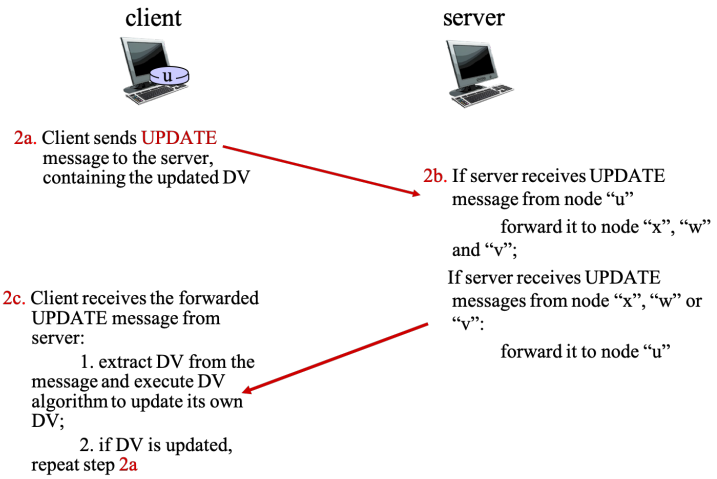


Figure 3: C-S Interactions During DV Execution

2.3 Implementation

- **Message Format.**You are free to design the format of `JOIN`, `UPDATE` and response messages. But for the `JOIN` message, it should contain `Router ID`, which could be either letter or decimal numbers. For the `UPDATE` and response messages, it should contain both `Router ID` and the list of tuples.
- **Socket Type.**You could use either **TCP or UDP** for the communication. UDP might be slightly easier than TCP.
- **Poisoned Reverse.**You do not need to consider this mechanism in your implementations.

3 Test and submission

The client needs to maintain a local forwarding table and propagate the results of the routing algorithm to this table. When the algorithm stablizes, you should print out this forwarding table. You could use the network topology shown in Figure 1 for testing.

Prepare a **README** file, containing instructions of compilation and execution of your code as well as your program environment, e.g., Python or Java version, Linux kernel version and any library dependencies. Submit it together with your **source code and configuration file** in a zip file.

NO LATE submissions will be accepted.