CS536 Lab2: Distance Vector Routing

Due Date: March 31, 2023 (23:59:59 PM), Total points: 50 points

1 Goal

In this lab, you are going to develop and use Distance Vector (DV) routing algorithm. Please work with a network topology with multiple nodes (routers) as the input and implement DV routing at every node in a distributed manner. You will further test your DV routing with different traffic settings and see how DV routing works with time-varying link costs.

2 Instructions

1. Read Chapter 5.2.2 and the lecture slides carefully, which will help you to understand how DV routing works.
2. Please write C codes that compile and operate correctly on the machines at HAAS G050.
   amber01.cs.purdue.edu
   amber02.cs.purdue.edu
   ...
   amber30.cs.purdue.edu

   Tip: you can develop and test with your lab programming at your local machine but finally make sure that it works at a machine at HAAS G050.
3. We provide a zip file called "lab2.zip" including
   - `main.c`: the starting source code;
   - `topo_4.txt`: the sample topology file (used in Part A, B and C);
   - `traffic_4.txt`: the sample traffic file (used in Part B and C);
4. Please start early.

2.1 Part A: Build A Network Simulator that Supports DV (25 points)

In this part, you need to build an Autonomous System (AS) with N routers (nodes) (assuming $N \leq 10$) with a **static** topology. Please start with your code from `main.c` provided in lab2.zip, which gives a network simulator framework. You need to implement DV routing protocol with a given static topology in the following steps.

1. **Input:** Your simulator should read a topology file (say, `topo.txt`) with a matrix $\{D_{i,j}\}, i, j \in \{0.., N-1\}$. N is the number of nodes (routers) and $D_{i,j}$ is the link cost from node i to node j. If nodes i and j are same, $D_{i,j} = 0$; If nodes i and j are directly connected (adjacent), $D_{i,j} = e$, where $e \geq 0$. Otherwise, nodes i and j are not directly connected and you should assign link cost $D_{i,j} = -1$.

   You can test your code with the topology shown in Figure 1 (Note that your code should work with any static topology with $N \leq 10$). Evidently, its corresponding topology file is given as follows (where $N = 4$):

$$\begin{matrix} 0 & 10 & 5 & 2 \\ 10 & 0 & 1 & -1 \\ 5 & 1 & 0 & 3 \\ 2 & -1 & 3 & 0 \end{matrix} \tag{1}$$

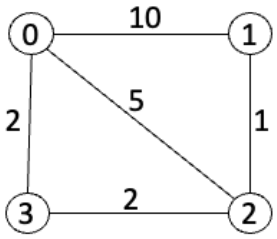

Figure 1: an example of topology and corresponding link costs.

Note that N nodes are marked as $0, 1, 2, \cdots, N-1$. So the link cost of node i to all the nodes $0, 1, 2, \cdots, N-1$ in order is given at the $i+1$th row of this matrix (the $i+1$th row vector). For example, the number "10" in the first row and the second column is the link cost between node 0 and node 1.

Please rename `main.c` into `mainA.c` and fill in the code to process the input file and load the topology for the following DV implementation. To run your code, please use the following command
`./mainA k_max [Input topo file path]`

where `k_max` is the maximum number of simulation slots and you should set it properly to make sure your algorithm converges. `[Input topo file path]` is the path to store the input topology file like "./topo_4.txt". For example, your command could be "`./mainA 50 ./topo_4.txt`".

Note: we will test your code with another static topology. You can assume that it is a connected graph and there is no need for your code to check whether it is a connected graph.

2. **DV Initialization.** You need to first implement DV initialization per node via

```
rtinit(struct distance_table *dt, int node_id, int *link_costs, int num_nodes)
```

For every node, `rtinit()` will be called only **ONCE** at the beginning of the simulation (k=0). For example, to initialize node 0, you need to use `rtinit(&dt, 0, link_cost, 4)` (Please check the sample code for more information). Node 0 should firstly initialize its distance table by setting all distances to -1. After that, node 0's distance vector in the distance table is changed to (0,10,5,2), representing the minimum distance from node 0 to all nodes in the topology. Lastly, node 0 needs to send its distance vector in a routing packet to its connected neighbors using `send2neighbor(rtpkt packet)` (you can find this function in the sample code) based on the input topology.

3. **DV Update.** You then need to implement DV update for each received routing packets. **Please assume that every DV update happens in a synchronous manner while it is not true for DV in reality.** DVs are updated at the start of each simulation slot and every updated DV will be sent out at the end of the simulation slot. There are no packet errors, losses or delays (more than one simulation slot) while distributing these routing packets containing DVs.

   Please use the following method to update DV for each node.

   `rtupdate(struct distance_table *dt, struct rtpkt recv_pkt)`
   This method will be called for each received routing packet at each simulation time slot $k \geq 1$. It parses the input, which is a routing packet *recv_pkt* to update the distance table based on the DV algorithm. If its own minimum distance to any of the other nodes is updated, the node informs its directly connected neighbors of this change in minimum cost by sending them a routing packet (`send2neighbor(rtpkt packet)`). Therefore, in the given example, nodes 0 and 1 will communicate with each other, but nodes 1 and 3 will not.

4. **Convergence.** Please make sure that your code can run successfully and produce correct and converged output under any topology with any ~~$N \geq 10$~~.

5. **Test cases and the expected outputs.** We provide one test case as shown in Figure 1 to test your implementation. Please check the sample topology file "`topo_4.txt`" in the lab2.zip. Also, please feel free to

create another network topology with any number of nodes for testing. (Note that the test cases used for the grading are different).

Assume the simulation starts at slot k = 0 for initialization and k≥ 1 when updating DVs at each simulation slot. You need to print out the current slot k followed by DVs of all the nodes in the ascending order *at the end of the simulation slots*. Please print them out in the first five slots (k = 0, 1, 2, 3, 4) and then every 10 slots (k = 10, 20, 30, $\cdots$) until they converge. In this test case, the expected output should be:

```
k=0:
node-0: 0 10 5 2
node-1: 10 0 1 -1
node-2: 5 1 0 3
node-3: 2 -1 3 0
k=1:
node-0: * * * *
node-1: * * * *
... ...
node-3: * * * *
k=2:  [more skipped]
... ...
```

Note for the current slot k, you need to print N+1 rows as the output. The first row is "k=?", where ? is the value of k, and the next N rows print out the distance vectors from node 0 to node N-1 in sequence, where each row uses "node-i: DV at node i to other nodes $0, 1, 2, \cdots, N-1$". Note that $*$ should be updated by your DV implementation. Please use `-1` to represent ∞. **Please make sure that there is only one space between two numbers and you cannot print any additional information in your submitted code. Otherwise, you will lose points.**

6. **Tip for Debugging.** For the debugging purpose, you can print more information on the console. However, please do not print such information when submitting your codes. So please add a debug flag to control whether to print the debug information. When the flag = TRUE, all your debugging messages will be printed; otherwise, only the above expected output above will be displayed (when the flag = FALSE). **You should make sure that the flag is set as FALSE in the codes you submit for this lab.**

2.2 Part B: Traffic Routing over a Static Topology (15 points)

Please continue this Part based on your Part A implementation (It is impossible to work on this part without finishing Part A). In this part, you need to use the forward table generated by the DV routing algorithm you developed to route traffic. Your code needs to accept network traffic as the input and get the routing paths for input traffic.

1. Copy `mainA.c` into `mainB.c`. Develop new features needed for this part in `mainB.c` (please do not revise your `mainA.c` as it is used to grade your Part A).

2. **Input:** Your simulator should read a topology file (`topo.txt`) as described above and a traffic file (`traffic.txt`) with a matrix $\{T_{i,j}\}, i \in \{0, .., V-1\}, j \in \{0, 1, 2\}$. V is the number of input traffic and $T_{i,:}$ represents the $(i+1)th$ traffic that should be sent at the end of each slot k, stating the source node id, destination node id and number of packets in sequence. Note that the traffic does not change over time.

   You can test your code with the topology shown in Figure 1 with the given traffic in "`traffic_4.txt`" (Note that your code should work with any static topology with $N \leq 10$ and any number of input traffic V). Evidently, the corresponding traffic file is given as follows (where $V = 4$):

$$\begin{matrix} 0 & 1 & 5 \\ 2 & 0 & 4 \\ 3 & 2 & 10 \\ 1 & 3 & 3 \end{matrix} \tag{2}$$

In this case, the *ith* row corresponds the *ith* traffic that should be sent at the end of each time slot k. For example, the first row indicates that for each time slot, there are 5 packets sent from node 0 to node 1. You can add more rows to create more traffic between different nodes.

Note: we assume that all the packets will reach their destination nodes within one time slot k no matter how long the path is and how many costs it might take.

To run your code, please use the following command:
`./mainB k_max [Input topo file path] [Input traffic file path]`

where `[Input traffic file path]` is the path to store the input traffic file like "`./traffic_4.txt`". For example, your command could be "`./mainB 50 ./topo_4.txt ./traffic_4.txt`"

3. **Routing.** You need to use the forwarding table determined by the current DV routing at each slot to determine how to route all the input traffic. We assume that the input traffic is sent out at the end of each slot after the DV is updated (done in Part A). Traffic routing starts at the first slot k =1.

   `route(int src_id, int dst_id, struct distance_table *dt)`

   This method will be called at each time slot $k \geq 1$. It inputs the source node id, destination node id and current node's distance table, and outputs the routing paths for the input traffic using the forward table generated by the DV routing algorithm. Note that, if there are multiple paths with the same costs between two nodes, the path with least hops is chosen.

4. **Convergence:** Please make sure that your code can run successfully and produce correct and converged output under any topology with any $N <= 10$ and V.

5. **Test cases and expected outputs:** The test case is same as the one in Part A and we provide an additional traffic file called "`traffic_4.txt`" in "lab2.zip" to support the routing feature in Part B. You can also create your own test cases, following the semantics of sample input files.

   You need to print out the current slot k followed by traffic information and routing path for every input traffic in the ascending order at the end of each slot. The traffic information is consisted of "source node id, destination node id and number of packets". Please print them out in the simulation slots between 1 and `k_max` (k = 1, 2, 3, 4, ..., `k_max`). In this test case, the expected output should be:

```
k=1:
0 1 5 0>2>1
2 0 4 2>3>0
3 2 10 3>2
1 3 3 1>2>3
k=2:
...
[skipped]
```

   Note that the output of each slot consists of one slot row (k=the value of k), followed by a traffic matrix which is exactly the same as the input V. At each traffic row, it is followed by its routing path. In this example, the traffic with 5 packets from node 0 to node 1 is routed through "node 0 → node 2 → node 1" and thus the print out is "0 1 5 0>2>1". **Please make sure that there are no space among the nodes along the path. Your submitted code cannot print any additional information on the console. Otherwise, you will lose points.**

2.3 Part C: Routing traffic under dynamic topology (10 points)

Please continue this Part based on your Part B implementation (It is impossible to work on this part without finishing Part B). In this part, you need to update the link cost based on the traffic along the link and then update DVs, which in turn impacts how to route traffic. We assume the topology connectivity is constant. If the link cost $D_{i,j}$ is negative at the beginning, it won't change over time (disconnected at all time). If the link cost $D_{i,j}$ is positive at the beginning, the link cost will be updated over time (here, the total number of packets over this link during the slot k).

1. Copy `mainB.c` into `mainC.c`. Develop new features needed for this part in `mainC.c` (please do not revise your `mainB.C` as it is used to grade your Part B).

2. **Input:** Your simulator should read a topology file (e.g., `topo_4.txt`) and a traffic file (, `traffic_4.txt`) described above. To run your code, please use the following command (the arguments are the same as Part B):
`./main`C `k_max [Input topo file path] [Input traffic file path]`

3. **Link Costs Update.** You need to write the codes to update link costs based on current traffic volume. At each simulation slot k, you run whatever you do in part B first and then update the link cost at the end of the slot. These updated link costs will be used to update DV at the next slot k+1. For example, if there is 10 packets sent from node 0 to node 1 and there is 10 packets sent from node 1 to node 0 during slot k (topo in Figure 1) the link cost between them should be updated to 20 at the start of the slot k+1.

4. **Test cases and expected outputs:** The test case is same as the one in Part B. You can also create your own test cases, following the semantics of sample input files.

   You need to print out the same output described in Part B, that is the current slot k and corresponding traffic information and routing path for every input traffic in the simulation slots between 1 and `k_max` (k = 1, 2, 3, 4, ..., `k_max`).

5. **Answer the question.** Have you observed the oscillation? Please run your Part C when k_max is small or large enough (for convergence). If yes, what do they look like? If no, please explain why not. Please answer this question in your report.

3 More Tips and Support

1. Please ask questions at PSO and/or Campuswire. If you have questions regarding your implementation, you can paste your screenshot of the code in the private channel to TAs on Campuswire. Note that posting your code in a public post at Campuswire will be treated as the violation against academic integrity.

2. If you want to use any late day, please send an email to cs536-ta@cs.purdue.edu and tell us. Please do not forget that each student can only use up to 3 late days for all the assignments.

3. You do not have to write a lot of comment details about the code but just adequately comment on your source code.

4. Please start early! There is no guarantee that you will get help within the last 24 hours prior to the submission.

4 Materials to turn in

Please submit your assignment on **Gradescope**.
Your should submit a zip file named as "Lab2_UID*.zip" including all the source files "`mainA.c, mainB.c and mainC.c`" and a report. For your report:

- Please include your name and student ID at the top of the first page.
- Please then include how to compile and run your source code (if the grader can't compile and run your source code by reading your report, the project is considered not to be finished).
- Include the answers as specified above in Part C.
- (Optional). You can post your printout for each part (all or partially) in the appendix of the report.