Programming Assignment 4: Networking Layer -> Routing -> Distance Vector Algorithm

Thanks to Rick Han @ Colorado State University for this problem.

The purpose of this programming assignment is to learn how distributed dynamic routing protocols like distance vector accomplish packet routing in practice. In this assignment, you will be asked to build a distance vector routing protocol that implements the distributed Bellman-Ford algorithm.

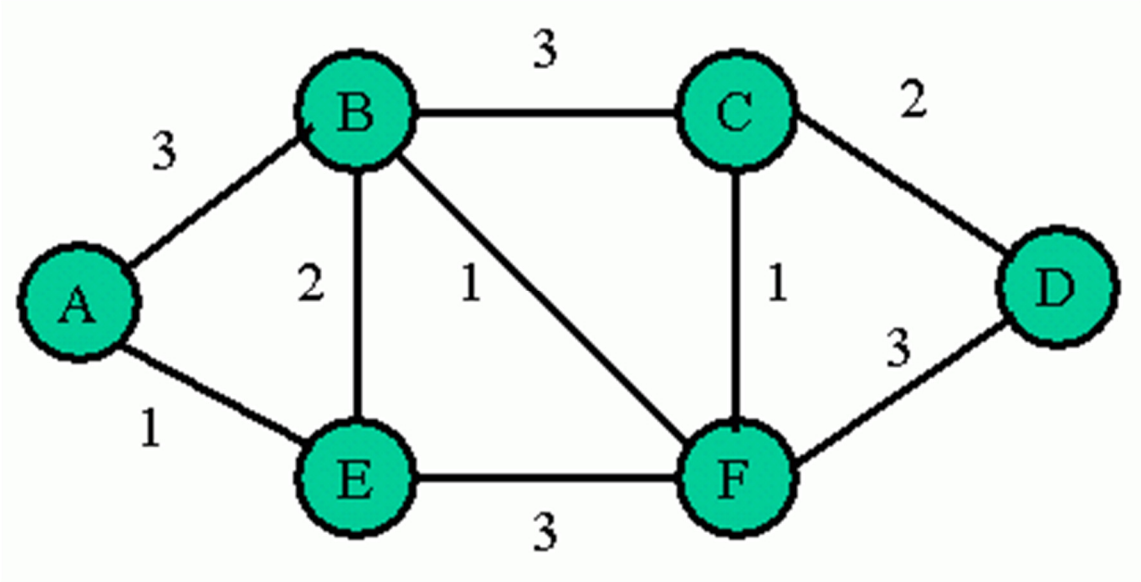

Figure 1 Network Topology

Figure 1 shows the topology of the network that you will be building, with nodes and link costs. Each router will be a separate process that communicates with other routers by exchanging UDP datagrams through the UDP socket interface. Routers should be implemented as Java threads.

Each of your routers will be identical, running the same routing code. You should be able to start your routers in any order. The router processes should then begin exchanging routing updates in the form of UDP-based distance vectors (DV) between neighbors. As each DV arrives, each router should execute again its shortest path algorithm and construct a routing/forwarding table. After many exchanges of routing updates, the routing tables should eventually converge to a stable state. You should be able to verify by inspection that the stable routing tables will route packets along the lowest cost paths.

In addition, your network of routing processes should also route actual data packets using the forwarding tables constructed at each router. These packets should follow the actual shortest path through the network as calculated by the distance vector algorithm.

(a) Building distance vector routing tables

You will be implementing distance vector (DV) routing using UDP processes. This is similar to how RIP is implemented, but differs in that you will be implementing *RIP-like* distance vector routing with different link costs. You will not be implementing an exact replica of RIP, since RIP assumes uniform link costs of one for all links. Your DV routing implementation will first need to exchange routing updates in the form of distance vectors between router processes that are neighbors. For simplicity and easier testing, let each DV router periodically advertise its distance vector to each of its neighbors every 5 seconds. For faster convergence, you can choose to speed up the DV advertisements to once per second, though you should document this in your README.

The distance vector advertised by router *i* should contain a list of destination nodes currently known by router *i* as well as the lowest cost paths to those currently known destination nodes. In the beginning, each router's first DV will list only the immediate neighbors of that router in the DV's list. As DVs are exchanged, each router's DV will grow. The resulting DV, after all destination nodes in the network topology have been discovered, will look like the following. You should include in the DV additional header information such as the ID of the node that generated this DV, the length of the DV, etc.

Destination	A	B	C	D	E	F
Cost	--	--	--	--	--	--

As each DV arrives at a router interface, your UDP router process should recalculate the routing table and DV based on the distributed Bellman-Ford Algorithm. That is, if you find from the just-arrived DV that a neighbor is advertising a shorter distance to a destination than you currently list in your routing table/DV, then you should update your routing/forwarding table accordingly. For example, suppose a router R currently lists in its routing table that the shortest path to destination S costs 20 through neighbor X. Suppose next that R receives a DV from neighbor Y advertising a distance cost of 10 to S. Also, suppose that the link cost between R and Y is 3. Then, router R should compute a distance cost of 13 to S through Y, which is less than its stored cost of 20 to S through X. Therefore, router R should update its routing table to reflect the new shortest path of 13 to S through Y. In case there are two routes with the same distance cost through different neighbors, then choose the neighbor with the lowest ID, i.e. A<B<C<D<E<F

For example, the initial routing/forwarding table at node A should look like:

Destination	Cost	Outgoing UDP port	Destination UDP port
B	3	10000 (Node A)	10001 (Node B)
E	1	10000 (Node A)	10005 (Node E)

The recommended UDP port #'s to use are shown in the figure below:

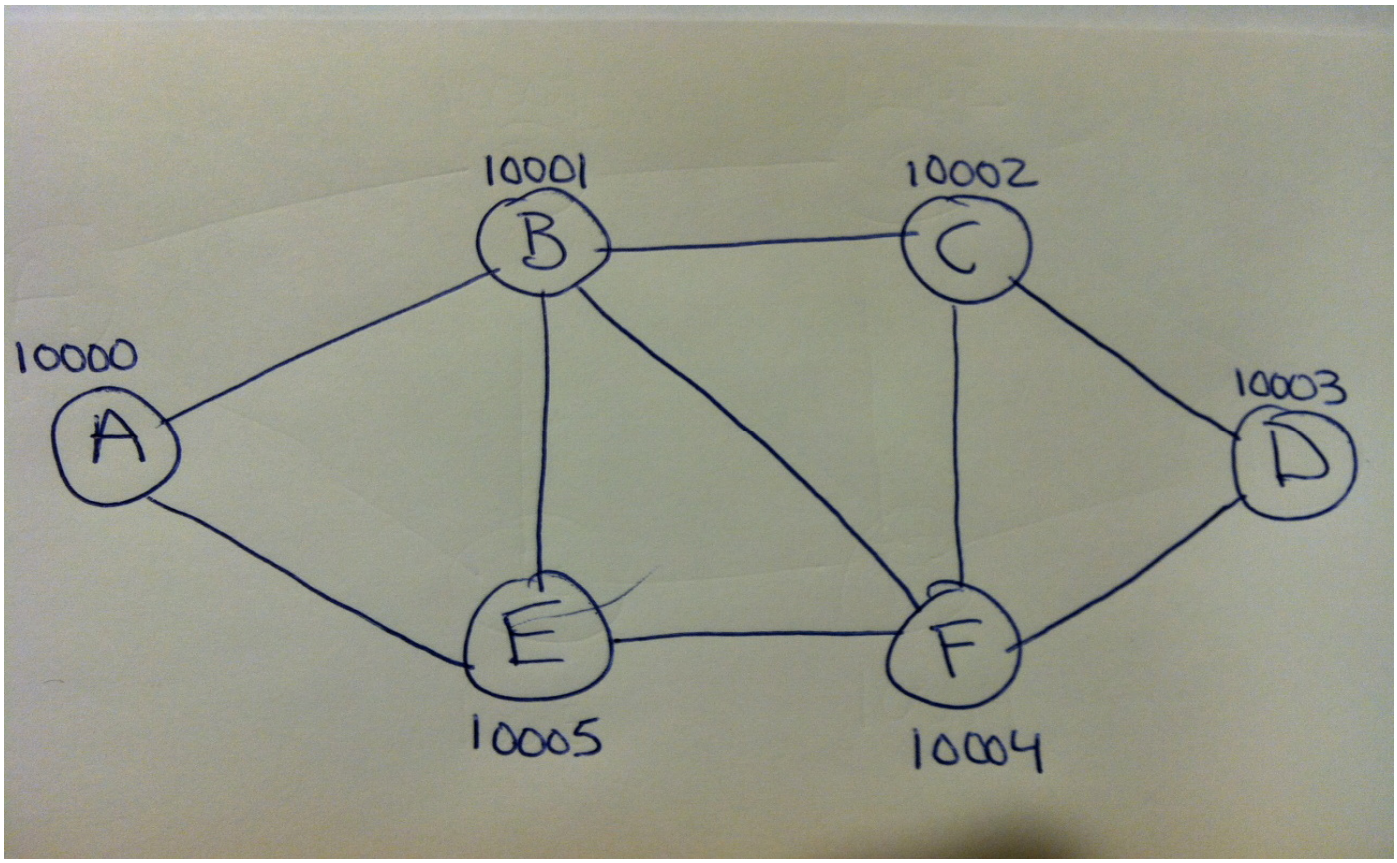

Figure 2 Network Addresses

As shown in Figure 2, each router should have a *distinct* UDP port. If you choose to test your routing implementation by placing at least one routing processes on a separate host, then you will only have to change the IP address of the remote router(s), not the port #. [Note: in an actual routing implementation, each routing interface will have a distinct IP address, rather than a distinct UDP port.] If you find that these port #'s are occupied on your machine(s), then you're free to modify the initialization file to use different distinct UDP port #'s above 5000. Please document this in the README.

For this assignment, each routing process should print out to an output file the routing table, but only when there has been a *change* in the routing table. If an arriving DV advertisement causes

a change in the routing table, then you should print out the timestamp, the routing table before the change, the DV that caused the change (including the neighbor it came from), and the new routing table. If an arriving DV does not cause a change in the routing table, then you do not need to print anything (though while you're debugging, you may wish to print out the routing table for each arriving DV). In the code that you hand in you should only print a routing table for those DV's that cause a change in the routing table.

Initialization

At startup, the neighboring nodes should find out about their immediate neighbors by reading the neighborhood topology information file. The format of each line of this file is a four-tuple:

<source router, destination router, source UDP port, link cost>

For example, the link between A and B contains the line <A,B,10000,3>. When router A starts up, it should read all lines of the initialization file where node A is listed as the source router. Also, router A should look at its neighbors' entries to find the UDP port where a DV packet should be sent. For example, for A to send to B, node A needs the UDP port # corresponding to B's interface with A, namely UDP port # 10001. This information is contained in the initialization file under the entries where A is the destination node, i.e. in the line <B,A,10001,3>.

Here's the file:

```
A,B,10001,3
A,E,10005,1
B,A,10000,3
B,C,10002,3
B,E,10005,2
B,F,10004,1
C,B,10001,3
C,D,10003,2
C,F,10004,1
D,C,10002,2
D,F,10004,3
E,A,10000,1
E,B,10001,2
E,F,10004,3
F,B,10001,1
F,C,10002,1
F,D,10003,3
F,E,10005,3
```

Your router should discover all nodes that are not immediate neighbors by exchanging DVs rather than reading directly from the initialization file (otherwise, that would defeat the whole purpose of this routing assignment!). For example, at startup, router A should only read information from the initialization file about neighboring routers B and E, i.e. their link costs and UDP send & receive port #'s. To find out about the existence of (and shortest paths to) nodes C, D, and F, router A must obtain this information by exchanging DVs. Your router is not allowed to obtain this information from the topology file. After many such DV exchanges, each router process will converge to the same list of all reachable nodes, though the distances to these nodes will differ depending on the router.

Command line

- You should create a file called routedDV.cpp, and create an executable called routedDV. When starting a router, simply pass in its name in the command line, e.g. "routedDV A" to start the router labeled "A". [In UNIX systems, RIP is typically implemented in a process called*routed*, namely a *route*r *d*aemon process.]

Output files

- Each routed_DV process should generate an output files containing print outs of the timestamped progression of routing tables for that router, as well as information about any data packets that have been routed through that router. Router A should produce an output file called routing_outputA.txt. Router B should produce one output file called routing_outputB.txt, etc.

(b) Routing data

After all routing tables have converged, which should occur after 30-60 seconds after the last router was started, then you should use your network of routers to route actual data packets from node A to node D. Your packet header should contain a type field that distinguishes packets of type "data" from packets of type "control" (containing the DV routing updates). Hence, data packets and control packets will arrive on the same UDP port numbers and socket buffers, and routers should look at the type field in each packet's header to determine whether the packet is data or a DV update. Your data packet can be any length, though for simplicity you could set it small at about 100 bytes to avoid fragmentation during the multi-hop routing. The data payload should contain a unique text phrase of your choosing. Please document your packet format, header fields, and payload phrase in your README.

Each router should forward the data packet towards the destination via the one neighbor that is along the shortest path to the destination, as determined by each node's routing table. Also, for each data packet that arrives at a router, you should print to the router's output file the timestamp, the ID of the source node of the data packet, the destination router, the UDP port in which the packet arrived, and the UDP source port along which the packet was forwarded. For the final destination D, the arrival of the data packet should trigger printing out of all the information above as well as the text phrase in the data payload.

The recommended way to inject a data packet into the network is to start up another router, call it host H, and have it send a data packet to router A with ultimate destination D. This is shown in Figure 3 below. For example, you could call "routedDV H", and then include a conditional if() clause in the code of routedDV.cpp such that if ID=H, then the code would send one data packet to UDP port 10000 of router A with ultimate destination D and then exits immediately. But before H exits, H should print out to its output file the data packet that it sent to destination D through router A, i.e. print out the header fields of the data packet such as the UDP destination port # on router A, the destination ID=D, and the text phrase in the data payload. The role of host H is simply to send one data packet and then exit, so H will not participate in DV routing. That is, if ID=H, then there is no need to execute the rest of the code in routed_DV.c corresponding to normal DV routing for routers A-F.

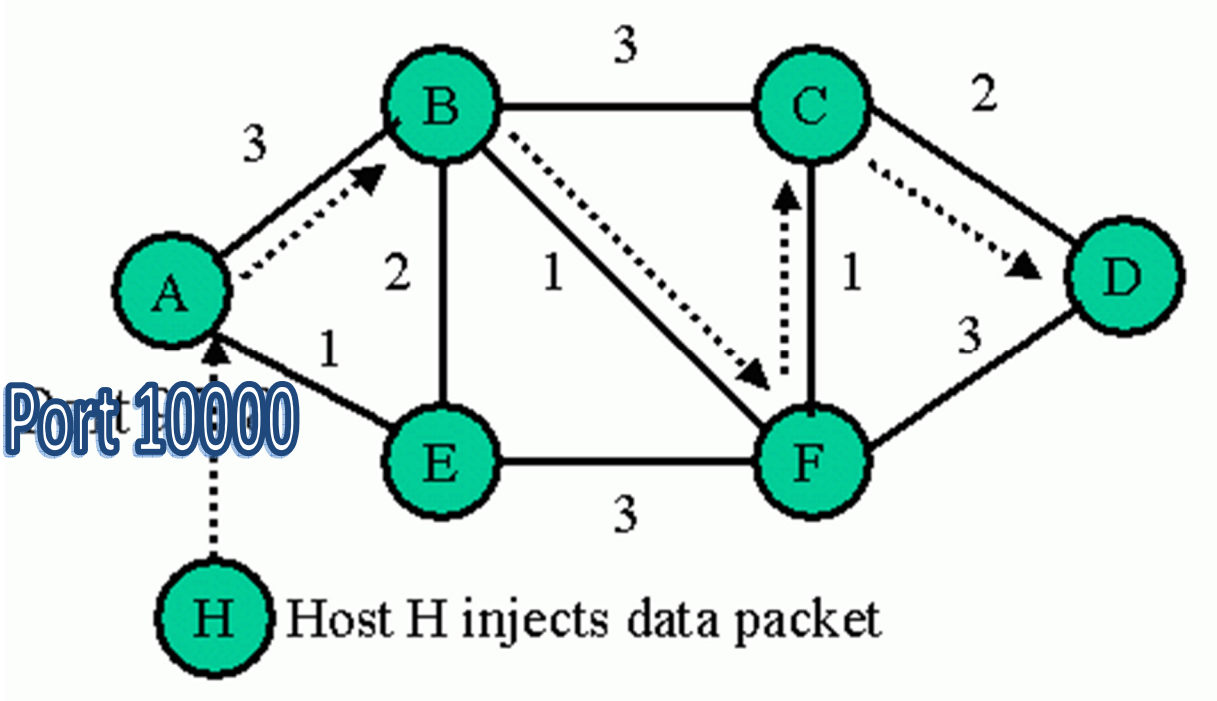

Figure 3 Host H injects a data packet into node A that is routed to destination D

(c) Adjusting to router failure

After your routing tables have converged, and after host H's data packet has been routed through your network from router A to destination D, then you should emulate a router failure by stopping the routing process corresponding to node C. Now follow again the sequence of events of parts (a) and (b), namely (a) wait for your routing tables to converge again to the new topology (and print out your routing tables as they're converging for routers A, B, D, E, and F) and then (b) after convergence, inject one more data packet into the new network topology by restarting host H and print out the resulting routing of the injected data packet (has it changed?).

A router process can recognize that a neighboring router process has failed by noting the absence of DV updates via a timeout. For simplicity, you should implement such a timeout for failure detection with a value that is some integer multiple of the periodic DV timeout. For example, if you chose a timeout of 5 seconds for DV updates, then you could decide that there has been a failure if you miss six DV updates in a row from a particular neighbor, i.e. choose a timeout for failure detection of 30 seconds. Please document your choice of failure timeout value in the README.

After a router receives its first DV from a particular neighbor, then it should set the failure detection timer. Since DV's are exchanged every 5 seconds, then the timer should be reset for each DV from a given neighbor, and should not timeout unless a neighboring router fails. If there is a timeout, then you should detect a router failure and update your routing tables

accordingly. You should also print out the routing tables in the same manner as if a DV had caused a change, i.e. print out the timestamp, old routing table, the ID of the failed neighbor, and the new routing table (even though it may be the case that the failed node does not cause the routing table to change, you should still print out all of the information requested).

After you've demonstrated that your network converges properly after a failure and can route a data packet correctly, then you're done for this programming assignment except for closing all output files cleanly. All routers including node C should stop cleanly and save their output files (C's closing of its output file will occur much earlier than the closing of the rest of the routers' output files). One way to signal to each router to exit and close its output file is to send a special exit packet to all routers A-F (not including C). For example, start a host X, using the call "routedDV X", and include a conditional if() clause in the code of routedDV.cpp such that if ID=X, then the code will send a UDP control packet to each router A-F (except C) and then exit. This will be a third type of "exit" packet, distinct from the DV updates and the data packets. Host X's one-time duty is to cleanly signal to all routers to exit. When a router receives an "exit" type packet on any of its UDP ports, then that router will close its output file and exit. The last point is how to cleanly close router C. We suggest that you incorporate logic in your code such that if ID=C and router C has just forwarded the first data packet from H towards D, then router C will automatically enter its exit mode, ignore all incoming packets, close its output file, and shut down after a delay of say 10-15 seconds. The exit packet can be sent to any of the UDP ports for a router. For example, router B has four UDP ports, and the exit packet from X can be sent to any one of the ports. You only need to send one exit packet to each router.