COM SCI 118 SPRING 2015

Project 2: Communication in the Internet of Things
Simple Distance Vector Routing Protocol in C/C++

1 Goal

The purpose of this project is to implement, validate and demonstrate a simple Distance Vector protocol of the type that might be used by Peer to Peer, Adhoc, Wireless Sensor or Internet of Things communication devices. You should use UDP Sockets and the C/C++ programming language to build a distance vector routing protocol that implements the distributed Bellman-Ford algorithm.

2 Instructions

1. In this project, you will be implementing a router that communicates with other routers by exchanging UDP datagrams through the UDP socket interface. A representative topology is shown in Figure 1 comprising six routers and associated link costs. In the real world these link costs may change as devices exhibit mobility, come under load, go offline etc.. To simplify your implementation these initial link costs are being provided.

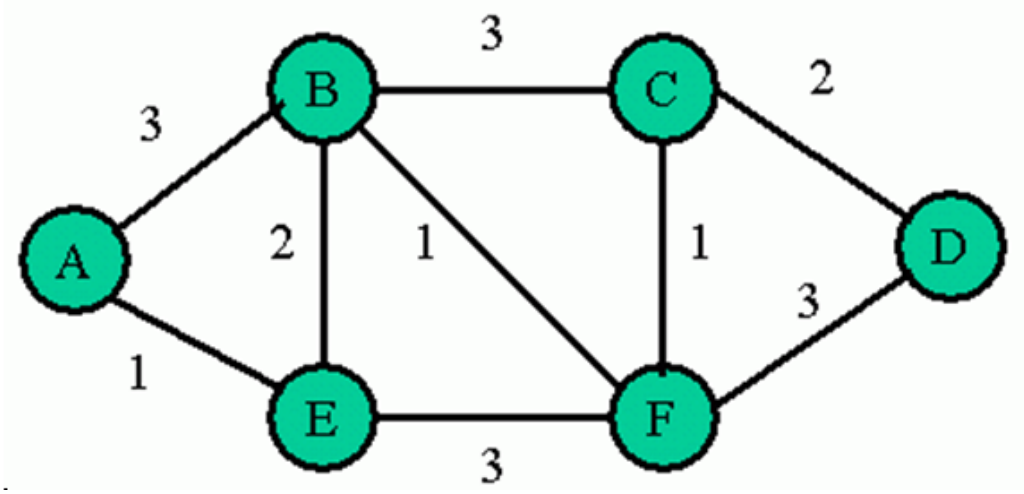

Figure 1: Figure 1 - Network Topology

2. In order to simplify the task somewhat, it will be possible to fully satisfy the requirements of this project by implementing a subset of core DV routing behaviours - specifically a subset no less than those implemented in TinyAODV. Your implementation may require additional functionality for proper operation - this is for you to decide and justify in your demo and report.

3. You must write one program implementing the router side. 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 again execute its shortest path algorithm and construct a routing/forwarding table. After many exchanges of routing updates, the routing tables should converge to a stable state.

4. You should be able to verify by inspection that the stable routing tables will route packets along the lowest cost paths.

5. For assessment purposes each routing process should print the routing table to an output file, 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.

6. You should also implement a configurable traffic generator that injects a data packet into the converged network and demonstrate that it successfully traverses the converged route. Your network of routing processes should also route this actual data packet using the forwarding tables constructed at each router. This packet should follow the actual shortest path through the network as calculated by the distance vector algorithm.

7. Finally you should demonstrate how the converged network responds to, and recovers from, router failure.

8. Only C/C++ code solutions are allowed in this project.

3 Approach

1. The best way to approach this project is in incremental steps. Do not try to implement all of the functionality at once.

2. Start of by getting UDP comms up and going between two endpoints. Decide on the message structure your are passing back and forward and get that going between the two endpoints. Then (for development purposes only) load the table entries provided below for nodes 1 and 2 into your two endpoints. Implement your core DV functionality, then change a link value in one of the tables and verify that the appropriate messages get generated and transmitted and that the receiver table updates appropriately. You now have all the core elements in place to complete the task.

3. Next extend your network to accommodate the node topology provided in Figure. Initialize ALL nodes (for development purposes only!) from the table below. Keep the network static. Now introduce a single DV change and watch how the network updates and converges. Satisfy yourself that everything is behaving correctly with udp messages, node updates and convergence behaviours across all nodes.

4. Next proceed to an implementation akin to TinyAODV with the following constraints: 1. RREP messages are only generated by the destination; 2. Routes never expire; 3. Only the hop count metric is used; 4. No messages are generated to keep routes active because routes never expire. Route errors are generated when a data message can no longer be sent over the path.

* You are eligible for full marks for this assignment upon i) successfully implementing, demonstrating and documenting the above behaviours, ii) implementing such additional functionality as may be required to satisfy the functional requirements set out herein, and iii) demonstrating a packet routing successfully across your converged network.

5. Extra Credit Activity: Extend your TinyAODV implementation as you see fit to more fully implement AODV features. Explain and justify your approach and decisions in your demo and report. Validated extra credit functionality can used to improve your grade in this project to a maximum of 100%

6. You should call your router file my-router.c (or .cpp as appropriate). You should create a makefile that will build your my-router process. You should provide a start-router script that initialises and starts the routers in a random order according to the topology provided. You should provide a script that calls your my-router binary with an appropriate command line to inject a data packet.

7. Each my-router process should generate 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.

8. Demonstration and Grading: At startup, the neighboring nodes should find out about their *immediate* neighbors (ONLY) 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 number corresponding to B's interface with A, namely UDP port number 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 a sample 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

9. 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 numbers's. In demo, and for grading purposes, router A must obtain information about the existence of (and shortest paths to) nodes C, D, and F, 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.

10. 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 upate . 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 report. 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.

11. The simplest way to create the traffic/packet generator will be to use your your existing router code and have it send a packet to (say) router A with ultimate destination D and then exit (see Figure 3). For example, you could call "myDV H", and then include a conditional if() clause in the code of myrouter.c 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 number 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 my-router.c corresponding to normal DV routing for routers A-F.

12. IMPORTANT: Where specific direction has not been provided as to a decision to make or path to follow, please make the decision that you think best enhances your overall solution and document the decisionand its rationale in your report.

4 Hints

1. For the internode communication phase you MAY find that using a third party asio library speeds your implementation - particularly concerning working with the UDP sockets on a asynchronous IO basis. I've used the boost library www.boost.org to good effect in the past. You don't have to use it and you shouldn't use it if you are not experienced with C++.

2. The recommended UDP port number's to use are shown in the 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 number. [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 one or more of these ports are already in use on your VM then you are free to change these port numbers but please document them in your report.

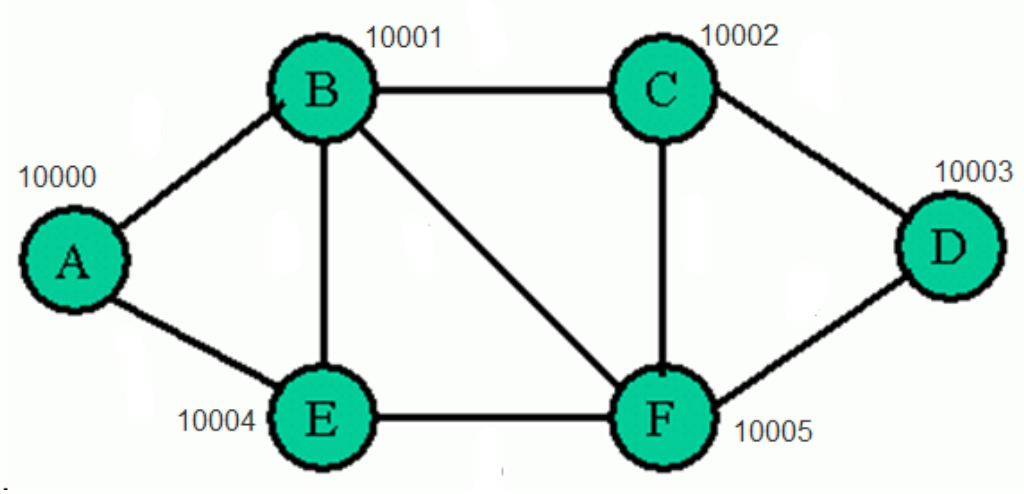

Figure 2: Figure 2 - Recommended Port Numbers

3. 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)

4. 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.

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

For informational and debugging purposes, 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.

5. 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 report.

5 Due Time and Demo

1. You are required to demo your program on Thursday(6/4) and Friday(6/5).
2. Submit a complete electronic copy of the project on courseweb on 11:59pm Wednesday(6/3).

5.1 Demo

In the demo, we provide you the provided VM machine. You then use make to compile your programs and run your my-router programs to demonstrate convergence. It is required by your program to print out the operations, which illustrate each step of the routing process, andassociate state and table information, on the screen. We may ask you to use different initial link costs to test your programs. We will ask you to explain how and why the routes arrived at are the least cost routes.

Demo Procedure:

1. Sign up for Demo : TA will distribute the signup sheet in the discussion section of the 8th week.
2. In Demo : You need to prepare 3 slides to present. 1 slide for design and implementation, 1 slide for experiences you gained, 1 slide for lessons learnt from project and suggestion for improvements to the project.
   - TA will ask you to demo the functionality step by step
   - TA will also ask you questions during demo, you need to answer the question clearly. All question will related to your project implementation.

6 Project Submission

1. Put all your files into a directory, must called $project2\_UID.tar$. UID is the student id of one of the students of the group.
2. Submit the file $project2\_UID.tar$ via SEAS online submission in course webpage.
3. The $project2\_UID.tar$ should contain the following files
   - Source codes (can be multiple files)
   - A report file (.doc or .pdf) no more than 3 pages, plus node routing tables. The report will contain:
     - Student names and Student IDs at the very beginning (**2 students per-group**).
     - Implementation description (structures, messages,routing/forwarding tables, etc).
     - Difficulties that you faced and how you resolved them.
   - Makefile
4. The TAs will only type "make" to compile your code, make sure your Makefile works in the provided VM machine.
5. Each group just needs to submit one set of files.