Programming Assignment: Unicast Routing

You have not submitted. You must earn 48/80 points to pass.

Deadline Pass this assignment by November 4, 09:59 PM PST

Instructions My submission

Programming Assignment 2: Unicast Routing

In this assignment, you will implement traditional shortest-path routing, with the

How to submit

When you're ready to submit, you can upload files

link state (LS) or distance/path vector (DV/PV) protocols.

The test setup will be the same environment as in MP1. Your nodes will use different virtual network adapters, with iptables restricting communication among them. We are providing you with the same script that our testing environment uses to establish these restrictions.

This MP introduces the unfortunate distinction between “network order” and “host order”, AKA endianness. The logical way to interpret 32 bits as an integer is to call the left-most bit the highest, and the right-most the lowest. This is the standard that programs sending raw binary integers over the internet are expected to follow. Due to historical quirks, however, x86 processors store integers reversed at the byte level. Look up htons(), htonl(), ntohs(), ntohl() for information on how to deal with it.

for each part of the assignment on the "My submission" tab.

programmingassignment2_files.zip

Routing Environment:

For this assignment, a virtual network will be emulated using the iptables in Linux. iptables is an application that allows users to configure specific rules that will be enforced by the kernel's netfilter framework. It acts as a packet filter and firewall that examines and directs traffic based on port, protocol and other criteria.

Your nodes will all run on the same machine. There will be a made-up topology applied to them in the following manner:

• For each node, a virtual interface (eth0:1, eth0:2, etc) will be created and given an IP address.

• A node with ID n gets address 10.1.1.n. IDs 0-255 inclusive are valid.

• Your program will be given its ID on the command line, and when binding a socket to receive UDP packets, it should specify the corresponding IP address (rather than INADDR_ANY / NULL).

• iptables rules will be applied to restrict which of these addresses can talk to which others. 10.1.1.30 and 10.1.1.0 can talk to each other if and only if they are neighbors in the made-up topology.

• The topology's connectivity is defined in a file that gets read by the script that creates the environment. The link costs are defined in files that tell nodes what their initial link costs to other nodes are, **if they are in fact neighbors**.

• A node's only way to determine who its neighbors are is to see who it can directly communicate with. Nodes do not get to see the aforementioned connectivity file.

Manager:

While running, your nodes will receive instructions and updated information from a manager program. You are not responsible for submitting an implementation of this manager; we will test using our own. Your interaction with the manager is very simple: it sends messages in individual UDP packets to your nodes on UDP port 7777, and they do not reply in any way.

The manager's packets have one of two meanings: “send a data packet into the network”, or “your direct link to node X now has cost N.” Their formats are:

“Send a data packet”:

"send"<4 ASCII bytes>**destID**<net order 2 bytes>**message**<some ASCII message (shorter than 100 bytes)>

Example: destination Id is 4 and message is “hello”. The manager

command will be:

“send4hello”

where 4 occupies 2 bytes and is in the network order. You need to convert it to the host order to get the correct destID. Note that there is no space delimiter among "send", destID and the message body.

“New cost to your neighbor”:

"cost"<4 ASCII bytes>**destID**<net order 2 bytes>**newCost**<net order 4 bytes>

Example: the manager informs node 2 that the cost to node 5 is set to 33. The command received by node 2 will be:

“cost533”

where 5 occupies 2 bytes and 33 occupies 4 bytes. Both of them are in the network order. You need to convert them to the host order to get the correct destID and newCost. Note that

there is no space delimiter among "cost", destID and newCost.

The source code of the manager is included in the attachment. You can read it to gain a better understanding of the format. Feel free to compile it and use it to test your nodes.

Your Nodes:

Whether you are writing an LS or DV/PV node, your node's interface to the assignment environment is the same. Your node should run like:

./ls_router nodeid initialcostsfile logfile

./vec_router nodeid initialcostsfile logfile

Examples:

./ls_router 5 node5costs logout5.txt

./vec_router 0 costs0.txt test3log0

When originating, forwarding, or receiving a data packet, your node should log the

event to its log file. The sender of a packet should also log when it learns that the packet was undeliverable. The format of the logging is described in the next section. **Note**: even if the node has nothing to write to the log file, the log file should still be created.

The link costs that your node receives from the input file and the manager don't tell your node whether those links actually currently exist, just what they would cost if they did. Your node therefore needs to constantly monitor which other nodes it is able to send/receive UDP packets directly to/from. In the attachment (main.c and monitor_neighbors.h), we have provided the code that you can use for this.

That concludes the description of how your nodes need to do I/O, interact with the manager program, and stay aware of the

topology. Now, for what they should actually accomplish:

• Using LS or DV/PV, maintain a correct forwarding table.

• Forward any data packets that come along according to the forwarding table.

• React to changes in the topology (changes in cost and/or connectivity).

• Your nodes should converge within 5 seconds of the most recent change.

Partition: the network might become partitioned, and your protocols should react correctly: when a node is asked to originate a packet towards a destination it does not know a path for, it should drop the packet, and rather than log a send event, log an unreachable event (see File Formats section).

Tie breaking: we would like everyone to have consistent output even on complex

topologies, so we ask you to follow specific tie-breaking rules.

• DV/PV: when two equally good paths are available, your node should choose the one whose next-hop node ID is lower.

• LS: When choosing which node to move to the finished set next, if there is a tie, choose the lowest node ID.

File Formats

Your nodes take the "initial costs file" as input, and write their output to a log file. The locations of both of these files are specified on the command line, as described earlier. The initial costs files might be read-only.

Initial costs file format:

<nodeid> <nodecost>

<nodeid> <nodecost>

Example initial costs file:

5 23453245

2 1

3 23

19 1919

200 23555

In this example, if this file was given to node 6, then the link between nodes 5 and 6 has cost 23453245 – as long as that link is up in the physical topology.

If you don't find an entry for a node, default to cost 1. We will only use positive costs – never 0 or negative. We will not try to break your program with malformed inputs. Once again, just because this file contains a line for node n, it does NOT imply that your node will be neighbors with n.

Log file: (See our provided code for sprintf()s that generate these lines correctly.)

Example log file:

forward packet dest [nodeid] nexthop [nodeid] message [text text]

sending packet dest [nodeid] nexthop [nodeid] message [text text]

receive packet message [text text text]

unreachable dest [nodeid]

...

In this example, the node forwarded a message bound for node 56, received a message for itself, originated packets for nodes 11 and 12 (but realized it couldn't reach 12), then forwarded another two packets.

forward packet dest 56 nexthop 11 message Message1

receive packet message Message2!

sending packet dest 11 nexthop 11 message hello there!

unreachable dest 12

forward packet dest 23 nexthop 11 message Message4

forward packet dest 56 nexthop 11 message Message5

Our tests will have data packets be sent relatively far apart, so don't worry about ordering.

Extra Notes:

To set an initial topology, run perl make_topology.pl thetopofile (an example topology file is provided). Note that you need to replace all the "eth0" in the make_topology.pl file to your VM's network name (e.g., enp0s3) to make the iptables work. This will set the connectivity between nodes. To give them their initial costs, provide each with an initialcosts file (examples of these are provided as well. See the spec document for more details on all file formats). You can just give them all an empty file, if you want all links to default to cost 1.

To make the nodes send messages, or change link costs while the programs are running, use manager_send.

To bring links up and down while running:

e.g. to bring the link between nodes 1 and 2 up:

sudo iptables -I OUTPUT -s 10.1.1.1 -d 10.1.1.2 -j ACCEPT ; sudo iptables -I OUTPUT -s 10.1.1.2 -d 10.1.1.1 -j ACCEPT

To take that link down:

sudo iptables -D OUTPUT -s 10.1.1.1 -d 10.1.1.2 -j ACCEPT ; sudo iptables -D OUTPUT -s 10.1.1.2 -d 10.1.1.1 -j ACCEPT

log file format:

sprintf(logLine, "sending packet dest %d nexthop %d message %s\n", dest, nexthop, message);

sprintf(logLine, "forward packet dest %d nexthop %d

message %s\n", dest, nexthop, message);

sprintf(logLine, "receive packet message %s\n", message);

sprintf(logLine, "unreachable dest %d\n", dest);

... and then fwrite(logLine, 1, strlen(logLine), theLogFile);

If that's all you do to the log file, you will have the correct format. (Don't worry about closing it; when your process gets killed, any output should be flushed + the file will be closed).

You can use the provided files as a starting point for the routing protocol implementation. However, it is fine if you want to start from scratch.

Grading

The grading will be determined by 8 tests, which are supposed to

be progressively harder and stress different elements of your code.

• Send a message to a non-neighbor node in a 3 node topology

• Send a message (by shortest path, of course) to a non-neighbor in a certain small (<20 node) topology.

• Send a message to a non-neighbor node in a certain large (>50 node) topology

• Switch to a better path when one becomes available in a certain large topology

• Correctly fall back to a worse path on a certain small topology

• Correctly report a failed send in a certain small topology that gets partitioned

• Get a packet through after a former partition is healed, large topology

• Converge within the time limit after a major, rapid series of changes

to the network (The time limit starts when the final change is made).

Grading Script:

Run the following grading script according to the instructions provided along with it and submit your results.

MP2_gradingscript.zip

-Copy your compressed code files (compressed into "code.zip") into the grading script directory. There is no format requirement for the code files. We keep the code for plagiarism check.

-Copy your compiled program in the grading script directory.

-Figure out the name of the network the router will run on. For the routing to work smoothly make sure your VM is connected to only one network. You can use the "sudo ifconfig" command to see the name of the network.

-Run the grading script as follows:

"python autograde.py [program_name] [network_name]"

Example: if program_name is "vec_router" and the network_name is "enp0s3", the command will be:

"python autograde.py vec_router enp0s3"

-After the successful execution of the grading script, a zip file named out.zip will be created with your program results. Submit this file on the assignment submission page on coursera.

NOTE: Before running the grading script, make sure your program works correctly with the example topology provided in the programming assignment files.