COMP/ELEC416 – Computer Networks
Project 3

Due: January 11, 2021 @ 11:59pm (Late submissions will not be accepted).

Submission of the project deliverables is via Blackboard.

Note: This is an individual project.

Network Layer Analysis and Routing Simulator

This project is about the **network layer** of the Internet protocol stack. The objectives are to examine the network layer data, the principles behind network layer services, and routing (path selection). Through this project, you will practice with Wireshark as well as a simplified network routing simulator.

The first part of the project requires you to analyze traffic data at the network layer through Wireshark. ICMP traffic will be monitored resulting from the application of a) *ping* and b) *traceroute* commands. The second part of the project involves working with the provided routing simulator and implementing routing strategies to analyze their performance.

Part I: ICMP Analysis

ICMP is a companion protocol to IP that helps IP to perform its functions by handling various error and test cases. The Internet Control Message Protocol (ICMP) is a supporting protocol in the Internet protocol suite. It is used by network devices, including routers, to send error messages and operational information indicating, for example, that a requested service is not available or that a host or router could not be reached. ICMP differs from transport protocols such as TCP and UDP in that it is not typically used to exchange data between systems, nor is it regularly employed by end-user network applications (with the exception of some diagnostic tools like ping and traceroute).

1.a: Ping Analysis

For this part of the project, you will use the ping command to analyze working of ICMP.

ping uses the ICMP protocol's mandatory ECHO_REQUEST datagram to elicit an ICMP ECHO_RESPONSE from a host or gateway. ECHO_REQUEST datagrams (``pings'') have an IP and ICMP header, followed by a struct timeval and then an arbitrary number of ``pad'' bytes

used to fill out the packet. ping works with both IPv4 and IPv6. Using only one of them explicitly can be enforced by specifying -4 or -6.

- Run the Wireshark packet capture.
- Ping the hostname URL assigned to you.
- You are required to send exactly 5 ping messages at 5 different times in a day or duration of the project and share the screenshot of the command prompt with a summary of ping completion and associated statistics.
- After the completion of ping, stop capture and answer the following questions. Remember to attach screenshots with each answer highlighting the relevant area.

1. What are the three layers in the ICMP packet?
2. What is TTL and its significance? Which layer does it reside in and is it constant (format and no. of bits wise) across IPV4 and IPV6 ping commands?
3. Why is it that an ICMP packet does not have source and destination port numbers?
4. What is the length of the datafied of the ICMP part Type -8? Elaborate on the structure of the datafield citing any correspondingly common and changing parts across various messages? If part of the data field, what do you think is the reason for that?
5. Find the minimum TTL below which the ping messages do not reach your particular URL destination.
6. How do the Identifier and Sequence Number compare for successive echo request packets?

1.b : Traceroute Analysis

In this part, you will use traceroute (may need to be installed) to perform the same set of actions as in Part-1.a.

Traceroute is implemented in different ways in Unix/Linux/MacOS and in Windows. In Unix/Linux, the source sends a series of UDP packets to the target destination using an unlikely destination port number; in Windows, the source sends a series of ICMP packets to the target destination. For both operating systems, the program sends the first packet with TTL=1, the second packet with TTL=2, and so on. Recall that a router will decrement a packet's TTL value as the packet passes through the router. When a packet arrives at a router with TTL=1, the router sends an ICMP error packet back to the source. In the following, we'll use the native Windows tracert program. A shareware version of a much nicer Windows Traceroute program is pingplotter (www.pingplotter.com).

The source and destination IP addresses in an IP packet denote the endpoints of an Internet path, not the IP routers on the network path the packet travels from the source to the destination. Traceroute is a utility for discovering this path. It works by eliciting ICMP TTL Exceeded responses from the router 1 hop away from the source towards the destination, then 2 hops away from the source, then 3 hops, and so forth until the destination is reached. The responses will identify the IP address of the router. Since traceroute takes advantage of common router implementations,

there is no guarantee that it will work for all routers along the path, and it is usual to see " * " responses when it fails for some portions of the path.

- Start up the Wireshark packet sniffer, and begin Wireshark packet capture.
- Use traceroute with the same URL. (Note that on a Windows machine, the command is "tracert" and not "traceroute".)
- On Linux, force the traceroute command to send ICMP packets instead of the UDP packets. You may look for this information using '*man traceroute*' and choosing the appropriate flag.
- When the Traceroute program terminates, stop packet capture in Wireshark. At the end of the experiment, your Command Prompt Window should show that for each TTL value, the source program sends three probe packets. Traceroute displays the RTTs for each of the probe packets, as well as the IP address (and possibly the name) of the router that returned the ICMP TTL-exceeded message.

7. How long is the ICMP header of a TTL Exceeded packet? Select different parts of the header in Wireshark to see how they correspond to the bytes in the packet.
8. How does your computer (the source) learn the IP address of a router along the path from a TTL exceeded packet?
9. How many times is each router along the path probed by traceroute?
10. Within the tracert measurements, is there a link whose delay is significantly longer than others? The echo request packets sent by traceroute are probing successively more distant routers along the path. You can look at these packets and see how they differ when they elicit responses from different routers.

Part II: Routing Implementation

In this part, you are asked to implement greedy routing algorithms at the network control plane and answer related questions. You are provided with a simulator and two topologies. You will need to implement four different algorithms and observe their behavior under the given topologies.

The topologies given to you can be visualized as following:

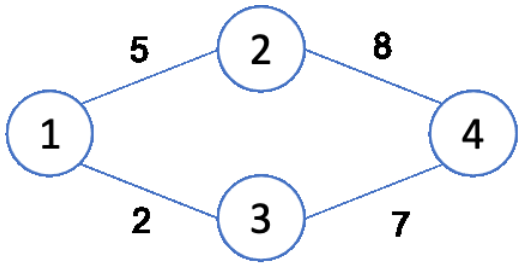

Fig 1. Topology 1 visualized.

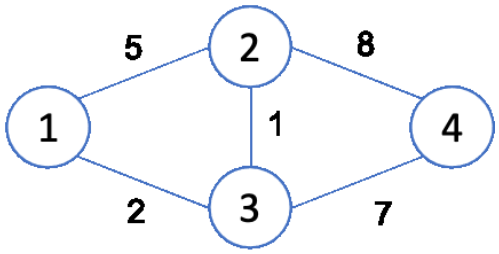

Fig 2. Topology 2 visualized.

The numbers on the nodes denote the address of the node (i.e., router), and the numbers on the edges denote the cost of that link. Please note that the graph is undirected. For example, node 2 has a link to node 3 with a cost of 1 and vice-versa. In our simulation, node 1 tries to send a packet to node 4.

Simulator Explanation

The simulator given to you can be executed by running the Main.java file. The outputs will be helpful to you in your implementation and answering the questions. The expected simulator outputs are given in the "expected_output.txt" file. The algorithms that you need to implement reside at the "algorithms" package. You will only need to implement the *selectNeighbors* method for each algorithm. Each node stores an instance of an Algorithm and invokes *selectNeighbors* method when there is a new packet to forward. The output of this method determines to which neighbors the packets will be forwarded to. Briefly, *selectNeighbors* takes the following parameters:

- **Origin**: The address of the origin of the packet.
- **Destination**: The address of the destination of the packet.
- **PreviousHop**: The address of the previous hop (i.e., the node that the packet was sent from.)
- **Neighbors**: A list of NeighborInfo instances that contain the neighbors that the node has.

Your implementation should select a subset of **Neighbors** and return them. The node that invoked the algorithm will route the packets to the list of neighbors that you have returned.

NaiveFloodingAlgorithm:

Go to NaiveFloodingAlgorithm class residing under the "algorithms" package. This algorithm is already given to you as an example. A node using this algorithm simply routes a packet to all of its neighbors (except the previous hop.) Run the simulator and observe the behavior of this algorithm. Then, answer the following questions in your report:

11. See that this algorithm succeeds in topology 1. What does the total communication cost represent, and why is it different from the path cost?
12. See that this algorithm fails in topology 2, and the simulator notes that the protocol does not converge. Why is this the case?

FloodingAlgorithm:
FloodingAlgorithm is a simple improvement over the NaiveFloodingAlgorithm: Each node in the topology only routes once. You can maintain a state, and return the neighbors only when *selectNeighbors* is called the first time. Go to FloodingAlgorithm class and implement it. Then, answer the following questions in your report:

13. Consider the path taken by the packet in topology 2 with this algorithm. Is this what you have expected? Why or why not?

NaiveMinimumCostAlgorithm:
NaiveMinimumCostAlgorithm always chooses the link with the smallest cost. This algorithm only returns a single neighbor (i.e., returns a list with only one neighbor,) as opposed to the flooding algorithms. While finding the appropriate neighbor, make sure that you do not consider the previous hop. Go to NaiveMinimumCostAlgorithm and implement it. Then, answer the following questions in your report:

14. If you implemented the algorithm as specified, it should succeed in topology 1 but fail in topology 2. Why does it fail in topology 2?

MinimumCostAlgorithm:
MinimumCostAlgorithm is an improvement over NaiveMinimumCostAlgorithm. More specifically, now we only consider the “edges” that were not previously used. This algorithm should maintain an exclusion set, i.e., a set of neighbors that should be excluded from routing to prevent cycles. When choosing the link with the minimum cost, we find the minimum of the neighbors that are **not** in the exclusion set. If all of the neighbors are already in the exclusion set, we simply choose a random neighbor. Please note that at the end of each call to *selectNeighbors*, two nodes should be added to the exclusion set: (1) the node that the packet was received from, (2) the node that the packet is being forwarded to. Go to MinimumCostAlgorithm and implement it.

15. List the nodes in the exclusion set of each node at the end of the simulation of topology 2.

Demonstration:

In the demo session, you are required to demonstrate the working of your Part-II implementation including the executions of the routing algorithms. You are also expected to answer questions on the concepts of Network Layer.
The dates and the schedule of the demonstrations will be announced later.

Project Deliverables:

Important Note: You are expected to submit a project report, in PDF format, that documents and explains all the steps you have performed in order to achieve the assigned tasks of the project. A full grade report is one that clearly details and illustrates the execution of the project. Anyone who follows your report should be able to reproduce your performed tasks without effort. Use screenshots to illustrate the steps and provide clear and precise textual descriptions as well. All reports would be analyzed for plagiarism. Please be aware of the KU Statement on Academic Honesty.

The name of your project .zip file must be <surname>-<KUSIS-id>.zip
You should turn in a single .zip file including:

- Source codes: Containing the source codes of your completed version of part-II.
- <surname>-<KUSIS-id>_P3.pdf titled Project Report.
    - For Part-I, your report should include the answers to the questions and the corresponding Wireshark screenshots.
    - For Part-II, a brief explanation of the implementation of the requirements supported by code snippets and answers to questions from this part.
- Saved capture files from the Wireshark.

Figures in your report should be scaled to be visible and clear enough. All figures should have captions, should be numbered according to their order of appearance in the report, and should be referenced and described clearly in your text. All pages should be numbered, and have headers the same as your file naming criteria.

If you employ any (online) resources in this project, you must reference them in your report. There is no page limit for your report.

Good Luck!