Assignment 1

CS 544: Topics in Networks
Deadline: 11:59 pm, 25 February 2020

Things to note before you start:

- No extensions in submission are allowed. Delay in submission will lead to penalty in marks.
- The programs can be written in C/C++/Java.
- Assignments submitted before the deadline will only be considered for evaluation.
- Submissions should be done via Moodle only. Please do not email your assignments separately to the TAs, it will not be considered for evaluation.
- Please read the questions carefully.
- Your code will be checked for plagiarism. Any kind of academic dishonesty, plagiarism, etc. will lead to penalties.
- NO sharing of code between students, submission of downloaded code (from the Internet, Campus LAN, or anywhere else) is allowed. The first instance of code copying will result in ZERO marks for the assignment. The second instance of code copying will result in a 'F' grade. Students may also be reported to the Students Disciplinary Committee, which can impose additional penalties.
- Please protect your Moodle account password. Do not share it with **ANYONE**, including your teammate. Do not share your academic disk drive space on Campus LAN.

1. The purpose of this part of the assignment is to understand link-disjoint routing algorithms and Virtual Circuit Switching concepts.

Inputs

The command line will specify the following:

- topology of the network
- connections between nodes

- routing tables
- forwarding table
- path taken from one node to another

For e.g., the input can be given as below:

$\%\ ./routing -top\ topology\ file -conn\ connections file -rt\ routingtable file\ -ft\ forwardingtable file\ -path\ paths file\ -flag\ hop|dist\ -p\ 0|1$

- The topology file contains on the first line: NodeCount (N), Edgecount (E) of the network. Nodes are numbered from 0 through $N-1$. On each subsequent line, there are four numbers – the first two integers represent the two endpoints of a bi-directional link, the third integer denotes the link's propagation delay (in ms), and the fourth integer denotes the link's capacity (in Mbps).
- The connections file contains on the first line: Number of Connection Requests (R). Connections are numbered from 0 through $R-1$. On each subsequent line, there are 5 numbers – the first two integers represent the source and destination node of a unidirectional connection, the remaining 3 integers denote the connection's stated capacity (in Mbps). The stated capacity of a connection request i specifies the requested bandwidth using three integers: $(b_i^{min}, b_i^{ave}, b_i^{max})$, which specify the minimum, average, and maximum bandwidth needed, respectively. Please note that there may be several connections between the same source-destination pair.

Routing

The program will first determine two shortest cost (not necessarily link-disjoint) paths for all node-pairs. You may use either hop or distance metric. The command line parameter will specify the choice.

Connections

The program will then process the specified set of connection requests.
Optimistic Approach: This is used if $-p$ command-line argument has value 0. Let C_l denote the total capacity of a link. Let $b_i^{equiv} = min[b_i^{max}, b_i^{ave} + 0.25 * (b_i^{max} - b_i^{min})]$. A connection is admitted if the following condition is met, along each link of the path selected for connection i:

$$b_i^{equiv} \leq \left(C_l - \sum_{j=1}^{n} b_i^{equiv} \right)$$

where n denotes the number of existing connections sharing a given link (along the path selected for the given connection) and j denotes the index

of a connection assigned on this link.
Next, for each source-destination pair, select the two link-disjoint paths. If adequate bandwidth is not available along the first shortest-cost path, then the network attempts to set up the connection on the second shortest-cost path. If this also fails, then the connection is NOT admitted, i.e. it fails to meet the admission control test. Once an available path is identified, the connection will be set up along the path. This primarily involves setting up link-unique VCIDs for a given connection request along all links of the path, and updating the corresponding forwarding tables at each intermediate node along the path.

Pessimistic Approach: If $-p$ value in command-line argument is 1 then use this approach. Let C_l denote the total capacity of link l. A connection is admitted if the following condition is met, along each link of the path selected for connection i:

$$b_i^{max} \leq \left(C_l - \sum_{j=1}^{n} b_j^{max} \right)$$

where j denotes the index of a connection assigned on a given link along the path selected for the given connection. This gets repeated for the pessimistic approach.

Outputs

The *routingtablefile* will contain the routing table information for all the nodes. For each network node, the corresponding routing table (i.e. two link-disjoint paths from the given node to all other nodes) displayed with the following fields:

Destination Node	Path	Path Delay	Path Cost

Table 1: *routingtablefile* format

The $forwardingtablefile$ will contain the forwarding table information for all the nodes. For each network node, the corresponding forwarding table (for all established connections) will have following format:

Router's ID	ID of Incoming Port	VCID	ID of Outgoing Port	VCID

Table 2: *forwardingtablefile* format

pathsfile will contain only one line consisting of two integers which represent the total number of requested connections and the total number of admitted connections, respectively.

For each connection that is admitted into the network, the output format is as follows:

Connection ID	Source	Destination	Path	VCID List	Path Cost

Table 3: Output format for each connection. Sort it based on connection ID.

Grading

- Routing setup: 15 points
- Connections setup: 25 points
- Output and viva voce: 10 points

NOTE: Two example topology files are given. You may additionally use your own topologies. For the 14−node NSFNET and 24−node ARPANET topologies, compare the blocking probability for your own set of link capacities/traffic requests. Example files for 100 requests are given. Evaluation will be done on a different set of input files.

2. **Coronavirus Inspection at Airports**
The purpose of this assignment is to understand the concept of queues.

The issue of Coronavirus is getting worse day by day. Therefore, the international airports in different countries are taking some preventive safety measures for the passengers who are traveling from China. The Government of India has ordered all the airports in India to do a chemical inspection for the passengers traveling from China. The process involves thermal screening of each passenger. The arrival of each passenger is considered as an *event*. Once the inspection is done, the passengers are free to leave the airport.

Case A

For the process of chemical inspection, 3 special officers are appointed in Delhi airport. Passengers arriving from China have to wait in queues at any one of the 3 officers. Since many flights come via China and some are directly from China, the officers are supposed to serve any number of arriving passengers throughout the day. The passengers can join any one of the queues with equal probability and are served on a First-Come-First-Serve (FCFS) basis. The arrival of passengers follows Poisson distribution with a mean arrival rate of λ_1. The service time taken by officers follows exponential distribution with mean service rate of μ_1.

Simulate the above given scenario with your own inputs and generate the following outputs for each queue:

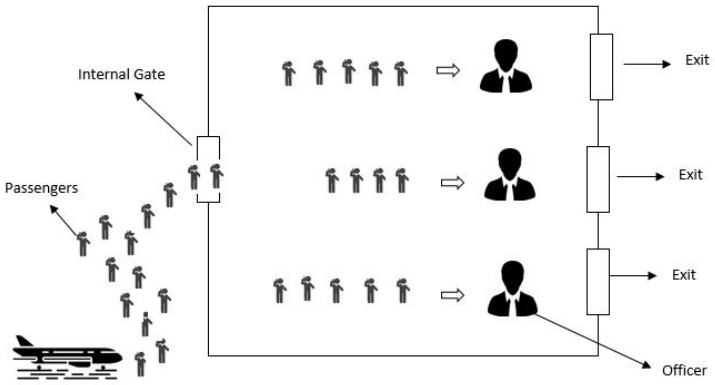

Figure 1: System scenario

- Average number of passengers getting inspected.
- Average response time for passengers in getting inspected.
- Average time for which a passenger has to wait until getting inspected.
- Average number of passengers waiting in queue before each officer (except the one getting inspected).

You can show the above in the console itself. Ensure that the program works for any input and output, verifies different test cases, and also throws exceptions, wherever needed.

Case B

On days when there are less number of passengers, only one queue is made, however, the number of serving officers remains the same. Passengers can go to any inspector with equal probability. Any number of passengers can stand in the queue but they are served on an FCFS basis. The arrival of passengers follows Poisson distribution with a mean arrival rate of λ_2. The service provided by officers follows exponential distribution in service time with mean service rate of μ_2.

Simulate the above given scenario with your own inputs and generate the following outputs:

- Average number of passengers getting inspected.
- Average response time for passengers in getting inspected.
- Average time for which a passenger has to wait until getting inspected.

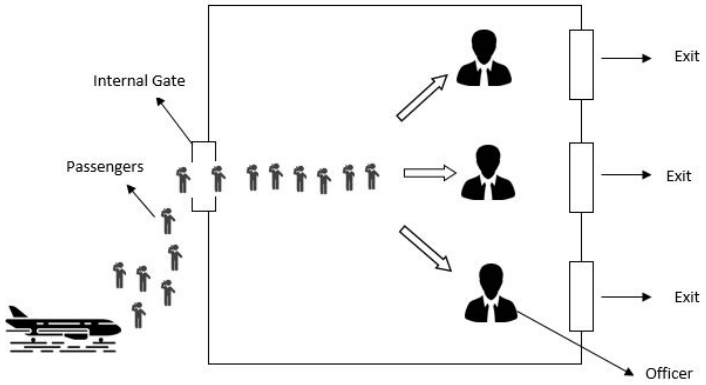

Figure 2: System scenario

- Average number of passengers waiting in queue before each officer (except the one getting inspected).

You can show the above in the console itself. Ensure that the program works for any input and output, verifies different test cases, and also throws exceptions, wherever needed.

Case C

For elderly passengers, special counters are made with chairs. Here too, 3 officers are there and each officer's counter has 10 chairs. The elderly passengers join one of the queues (of chairs) with equal probability and are served on an FCFS basis. If the queues are full, they join the queue with other non-elderly/regular passengers. The probability of such an event happening is 0.01. The arrival of passengers at each counter follows Poisson distribution with a mean arrival rate of λ_3. The service provided by each of the officers follows exponential distribution for the service time with mean service rate of μ_3. Simulate the above given scenario with your own inputs and generate the following outputs:

- Average number of passengers getting inspected.
- Average response time for passengers in getting inspected.
- Average time for which a passenger has to wait until getting inspected.
- Average number of passengers waiting in queue before each officer (except the one getting inspected).

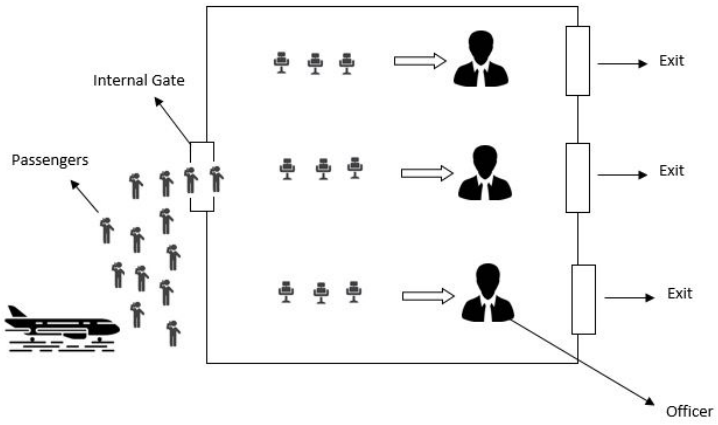

Figure 3: System scenario

You can show the above in the console itself. Ensure that the program works for any input and output, verifies different test cases, and also throws exceptions, wherever needed.

Grading

- Case A – 10 marks
- Case B – 10 marks
- Case C – 20 marks
- Output, comments, viva voce - 10 marks

NOTE: The above question is given for you to simulate a real life scenario and understand the concept of queues. Therefore, **DO NOT** use queuing theory formulas to get the results in your code. You can use them to verify the correctness of your code.

What to submit

A single .zip/tar.gz file containing

- Separate subfolders for each question.
- Source files, compiled executable.

- Example Input files and corresponding Output files
- Makefile
- README file which explains clearly, how to compile the program and run it. Mention in case there are any errors or bugs in the code.
- Name the .zip/tar.gz file you are submitting the format - "AssignmentNo.–Roll no. of Group members separated by underscore".