CS6040 – Router Architectures and Algorithms
Lab 1: Virtual Circuit Switching

July 2024 Semester, Prof. Krishna Sivalingam
Due date: Aug. 23, 2024, 11PM, On IITM Moodle

Assigned: Aug. 9, 2024

The purpose of this assignment is to understand Virtual Circuit Switching concepts.

1 Inputs

The command line will specify the following parameters:

```
% ./routing -top topologyfile -conn connectionsfile -rt routingtablefile \
 -ft forwardingfile -path pathsfile -flag hop|dist -p 0|1
```

- The topology file that contains on the first line: NodeCount (N), Edgecount (E) in the network.

  Nodes are numbered 0 through $N-1$.

  On each subsequent line, there are four numbers, where the first two integers represent the two endpoints of a bi-directional link; the third integer denotes the link's propagation delay (in *ms*), the fourth integer denotes the link's capacity (in Mbps).

- The connections file that contains on the first line: Number of Connection Requests (R).

  Connections are implicitly numbered 0 through $R-1$.

  On each subsequent line, there are 5 numbers, where the first two integers represent the source and destination node of a *unidirectional connection*, and the remaining 3 integers denote the connection's stated capacity (in Mbps), i.e. a connection request i specifies the requested bandwidth using three integers: $(b^i_{min}, b^i_{ave}, b^i_{max})$, which respectively specify the minimum, average and maximum bandwidth needed. Note that there may be several connections between the same source-destination pair.

2 Processing

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

You can design your own routing algorithm for computing the two paths. There are well-known shortest path algorithms for link-disjoint paths, etc.

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 l. Let $b^i_{equiv} = \min[b^i_{max}, b^i_{ave} + 0.35 * (b^i_{max} - b^i_{min})]$. A connection is admitted if the following condition is met, along each link of the path selected for connection i:

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

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: This is used if the `-p` command-line argument has value 1. Let C_l denote the total capacity of a 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. Repeat the above for the pessimistic approach.

3 Outputs

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

Destination Node	Path (from Source to Dest)	Path Delay	Path Cost

The *forwardingfile* will contain the forwarding table information for all the nodes. For each network node, the corresponding forwarding table for all established connections will be displayed with the following fields:

This Router's ID	Node ID of Incoming Port	VC ID	Node ID of Outgoing Port	VC ID

The *pathsfile* will first contain one line that has two integers: the total number of requested connections and the total number of admitted connections.

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

Conn. ID	Source	Dest	Path	VC ID List	PathCost

The above list will be sorted based on connection ID.

4 What to Submit on IITM Moodle

A single tar.gz file containing:

- Source files, compiled executable
- Example Input files used and Corresponding Output Files generated
- Makefile: The TA should be able to run 'make' from the command-line and then run the executable
- README File that explains how to compile and run the program; whether your programs works correctly or whether there are any known bugs/errors in your program.

- Technical Report: For the 14-node NSFNET and 24-node ARPANET topologies, compare the blocking probability for your own set of link capacities/traffic requests (100, 200 and 300 requests). The TAs will evaluate on a different set of input files.

  See:
  `https://en.wikipedia.org/wiki/National_Science_Foundation_Network`
  `https://en.wikipedia.org/wiki/ARPANET`

5 Grading

- Routing Setup: 35 points
- Connections Setup: 55 points
- Report and Viva Voce: 10 points

6 Policies

- The program can be written in C/C++/Java/Python.
- There will be no sample input / output files provided.
- This is an INDIVIDUAL assignment. Please refer to first day handout (on Moodle) regarding penalties for any form of academic dishonesty, plagiarism, etc. There should be no downloaded code.
- Software for checking plagiarism the code will be used.
- Please start on the assignment right away and write the program a few functions at a time – you can start working on implementing the input file parsing, routing algorithm implementation, etc.

  Starting to write the program just 1-2 days before the deadline is often not known to succeed.