Computer Networks Assignment 4

Prof. Prasad

Due date: Dec 5, 2025
Learning goals:

1. Students will be able to describe the role of the network layer.
2. Students will be able to identify, design and develop the data structures and methods required for routing and forwarding of packets.
3. Students will be able to write multithreaded Java code.

Java knowledge required: Read up on the following Java classes: Thread, Timer, TimerTask, Object, and the synchronised statement

---

The setup described in this file allows you to use Terminal tabs as routers or your network nodes. For getting a 3, you just need to get your code to work between Terminal tabs, but you are more than welcome to design and implement your code to work with multiple machines. Some preliminary guidelines for this design are given in the description of the project on theSpring to help you get started, but talk to the professor if you need more detailed guidelines.
To compile Java files, you run the command `javac *.java` which compiles all the Java files, and to run a particular Java file you say `java filename`, for example, `java Client`.

In Project 3, the client and server exchanged segments using a direct TCP connection between the two nodes. In Project 4, we will re-implement the data exchange using the simple network layer that implements the simple network protocol (SNP) instead. The network layer is responsible for routing and forwarding packets. The routing table is constructed using the link-state algorithm, which will determine the lowest cost routes from source to destination.

Routing and Next Hop Table

Create a NetworkNode.java class that stores the information such as node id and port and create a Graph class to hold the NetworkNode objects and the cost values given in the network.dat file. The Graph class should contain

- a method to read the network.dat file and load the router nodes and edge weights; you can assume an undirected graph, i.e., data flows in both ways,
- another method, which given a start and end node, calculates the shortest path from start to end node and set the next hop value for the end node in the next-hop table of the start node.

It is up to you to come up with this entire implementation. How do you use the shortest path method to find the next hop for each node? How do you store the ids and the ports for each node? A sample network.dat file contains both information about the nodes (their IDs and portnumbers) and the link costs:

```
node1 port1
node2 port2
```

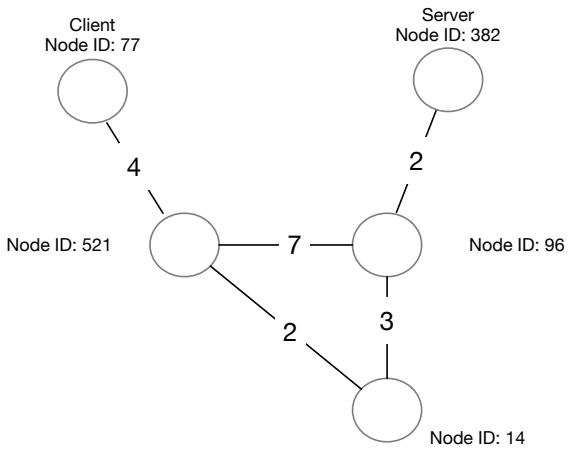

Figure 1: network

```
node3 port3
node4 port4
node5 port5
node1 node2 cost1-2
node2 node3 cost2-3
node2 node4 cost2-4
node4 node1 cost4-1
node4 node5 cost4-5
```

Some hints: When the Client and Server objects are constructed, it should use the Graph's Dijkstra method to create a nextHop table. Both the Client and Server objects should have NetworkNode objects as a field, a good implementation will simply use the methods in the NetworkNode and Graph classes in the Client and Server classes, while ensuring that the Graph class only contains graph-related methods, the Network Node includes all the network layer related methods, the Client only client-related methods and Server only server-related methods.

In our example network, the client is node 77 and server is node 382. Instead of connecting to the server, the client should instead open up a connection with the nextHop router instead, and similarly the server should be listening on for connections from the router it is connected to, so change your Client and Server code appropriately. To keep the code simple, you may hard-code the node ID and port numbers.

Data Transfer

The data transfer at the network layer involves the exchange of packets or datagrams, instead of segments. Instead of sending a segment, you will need to modify the Client.java class so that the client instead creates a packet that contains the segment, and send the packet instead on the socket. The segment's destination node ID, in this case, would be the server's node ID of 382. To send the packet, the client would retrieve the next hop node for the node corresponding to the server, create a socket to connect to the next hop node and send the packet to the next hop, instead of the server.

The router nodes, on receiving the packet, should quickly examine the destination id, and find the next hop's node ID from its next hop table – what data structure would make most sense for the next hop table?

The router then opens up a connection and sends the packet to the "next hop" router. When the server receives the packet, it checks the destination ID and on realizing it is the destination node should retrieve the segment – and continue with what you implemented in Asst 3.

The Packet class contains the following information:

```
int srcNodeID;   //src node ID
int destNodeID;   //dest node ID
Segment seg;    //a Segment object
```

To keep SNP simple, we will be implementing the centralized algorithm so you won't be using the multiple connections in the TCBServer and TCBClient, which had made more sense for a Distance Vector algorithm, which you could still implement if you wish to extend this assignment at your own time to do the decentralized distance vector Bellman-Ford algorithm.
Due to the virtual classes in Spring 2021, we will be using multiple terminals as the different nodes, so when you create the NetworkNode objects, you have to update only the port where the node is accepting connections since the ip address is going to be localhost. For the Client and Server classes, you may choose to add nodeID and port as fields; the port was hard-coded earlier, instead make sure you can get it from the network.dat file using the node ID. You should pass node ID as a command-line argument for all the nodes. So, you can run 'java NetworkNode 14' and obtain the node id as `Integer.parseInt(args[0])`, after checking if `args.length==1`; do the same for Client and Server also. You can get the ports of all the nodes from the network file.
For the setup shown in Fig 1, one possible network.dat file could be as follows:

```
77 4223
521 4534
14 5542
96 5146
382 4675
77 521 4
521 14 2
14 96 3
521 96 7
382 96 2
```

A few important things to keep in mind:

- In this case, a NetworkNode object should be able to both send and receive packets, so make sure this happens on different threads. Every time you send, listen for or receive packets, it should always be in a thread.
- Use Thread.sleep() wherever necessary to leave ample time for your messages to be received.
- To run the fourth assignment, use six Terminal tabs. Start the four router nodes using `java NetworkNode nodeID portnum`, start the Server on one tab, and finally, start the Client on the last tab. Feel free to let the main threads sleep for 15 seconds so that the nextHop tables are created for all the objects, before the Client reads the file and sends multiple packets. Make sure to add print statements in the NetworkNode objects so you can track the packets as they go through each of the intermediary nodes, before you obtain it at the Server.
- Test with a different network.dat file to make sure it still works when you change the port numbers and link costs.

Congratulations! You have built your mini Internet – this is a huge achievement!

What to submit:

Submit screenshots of all the six terminal tabs as well as the files Client.java, Server.java, TCB-Server.java, TCBClient.java, Segment.java, Packet.java, Graph.java, NetworkNode.java, and the network.dat file to theSpring.

Grading Rubric

Graph class and Dijkstra's algorithm	25 points
Design of the NetworkNode class and how it uses Graph class to create next hop table	25 points
Design of the Packet class	5 points
Updated Client which takes each segment from buffer, creates and sends Packet	10 points
Updated Server which receives Packet, obtains the segment and puts in receive buffer	10 points
NetworkNode node that receives packet, determines next hop and forwards packet to next hop	15 points
Documentation	10 points

25% points off for code that does not run, 40% off for code that does not compile.