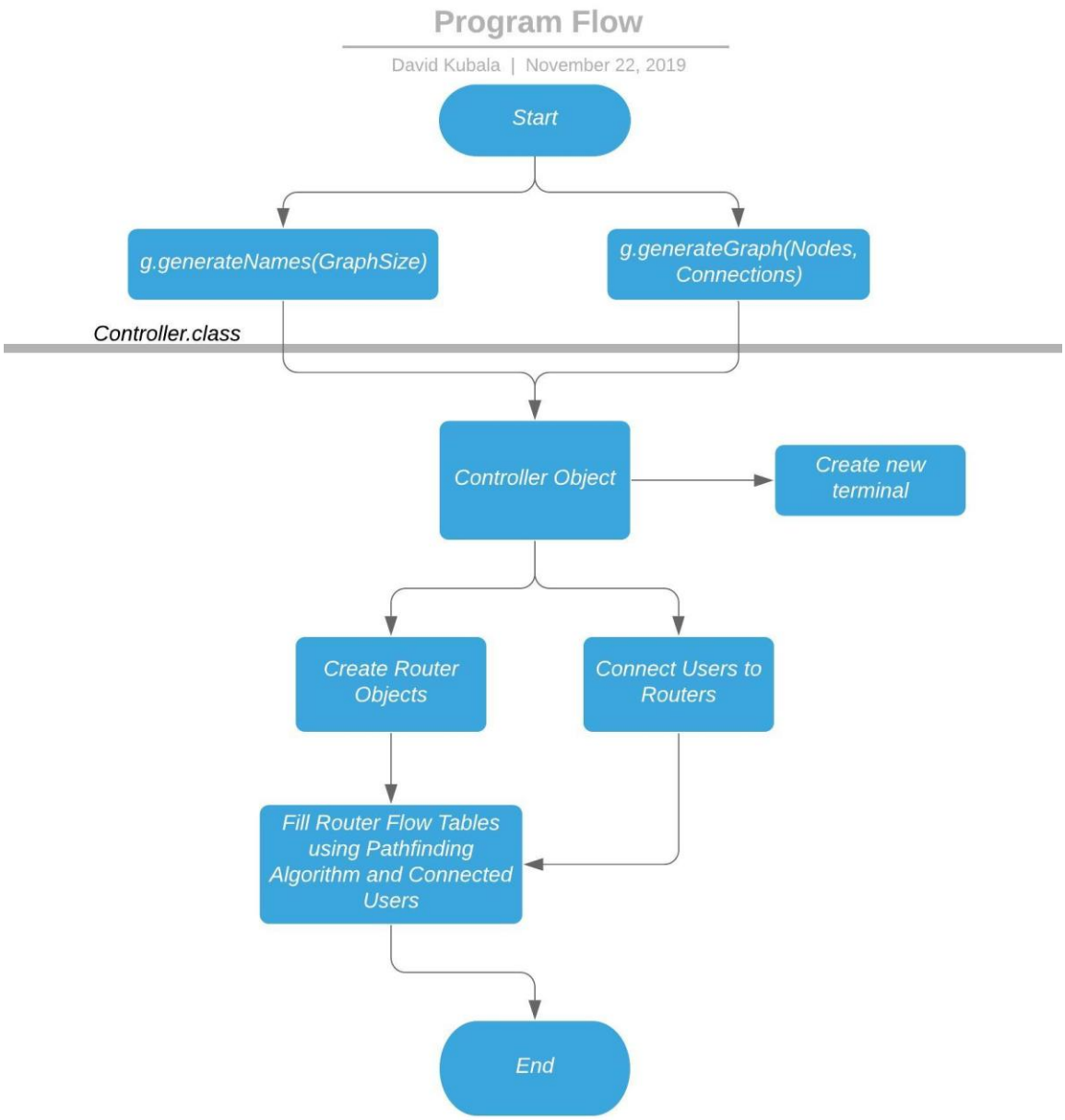
Trinity College Dublin

Telecommunications II

OpenFlow

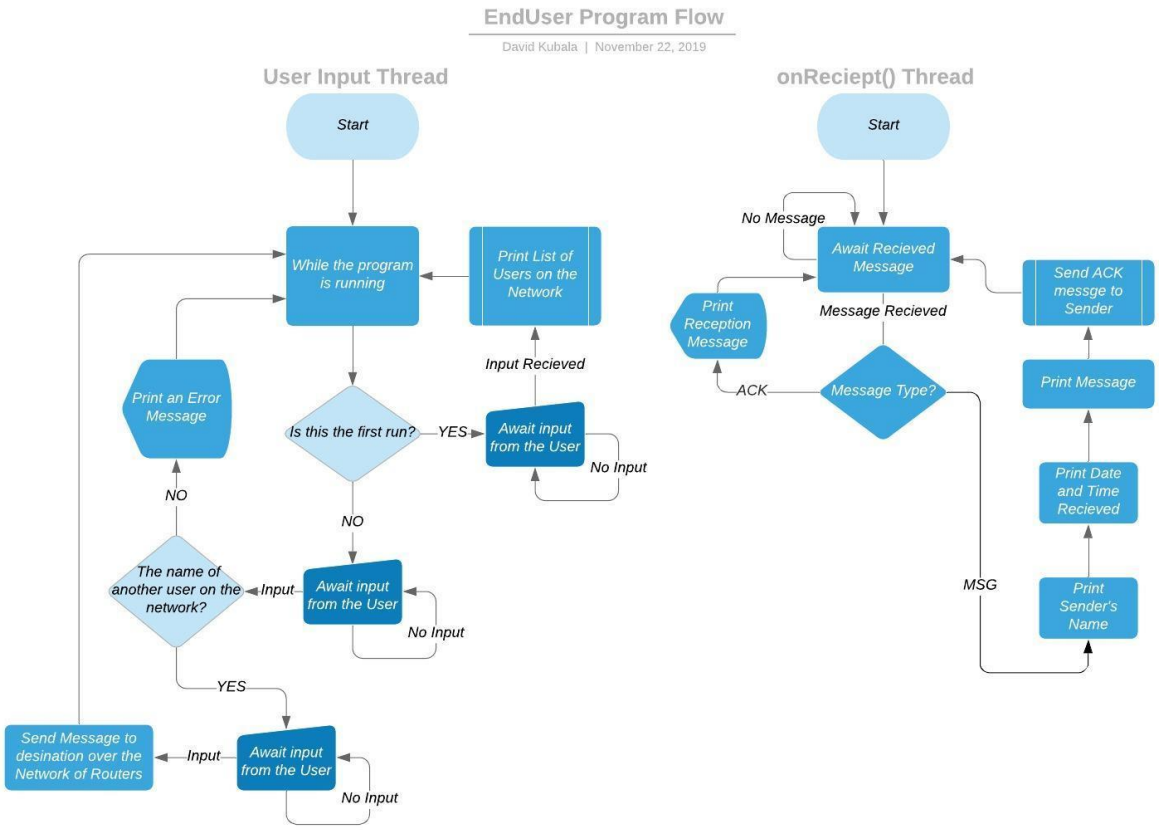
Trinity College Dublin
Thursday, November 21, 2019
David Kubala



Program Structure

MAIN CLASS - CONTROLLER CLASS

The program I've developed creates a random network using the GraphGenerator class, represented as a matrix. Using this randomly generated graph, the program first connects a router for each node. Next, the users are connected to the graph via a specified router. Once this is completed, the router uses the Pathfinder class to find the shortest distance over the network between the 2 specified nodes of each user. The flow tables are filled in for each router, and the program awaits user input from each separately-threaded user class.



EndUser

The EndUser class is very important in the flow of the program. Using the terminal line on each instance of the EndUser class is the only way the User can communicate with other users on the network.

Data

Terminal *t*
String *username*
int *connectedRouter*
int *port*
InetSocketAddress *adr*
ArrayList<> *NetworkUsers*

Methods

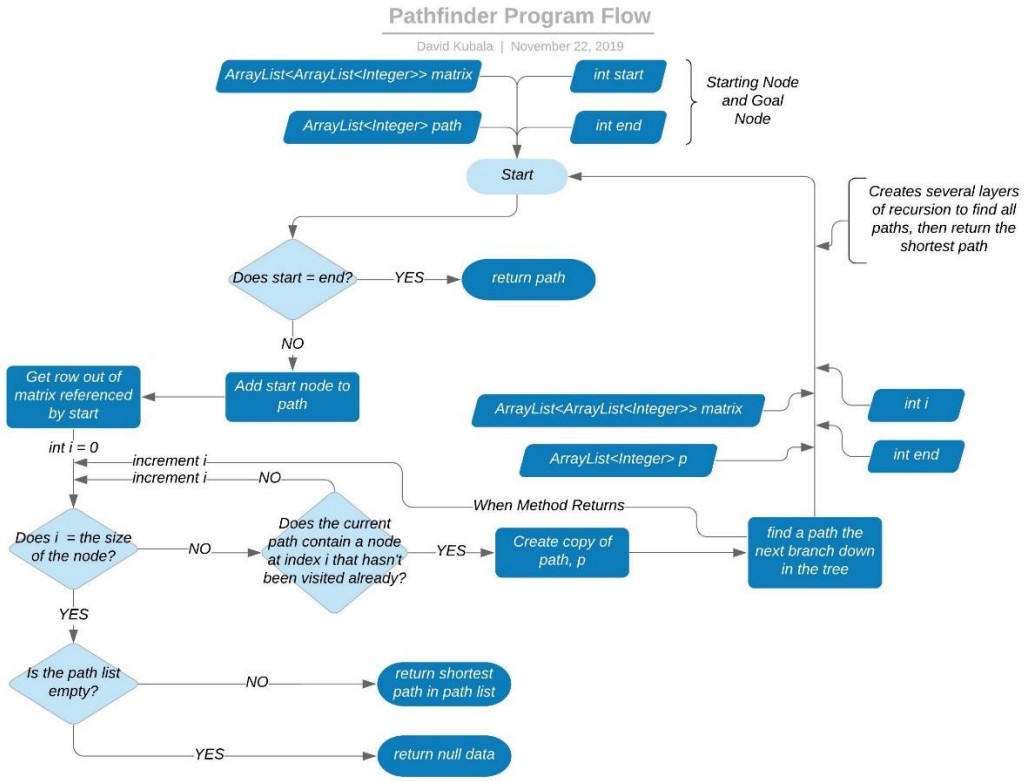
endUser()
constructor
onReciept()
checks if a new packet has been recieved
run()
checks if the user has added input to terminal
printUserList()
prints the network of users to the terminal

When a new instance of EndUser is created from the Controller class, a new terminal window is created to support the user's activities. The terminal acts as the endpoint (frontend) of the code to allow the user to interact with the program.

The EndUser class then starts two separate threads, continuing to run the run() method, as well as the onReciept() method.

The run() method is constantly scanning for user input on the terminal. Once input is added, it is checked, then sent along to the specified user.

The onReciept() method awaits new messages, and determines what to do with them based on if they are an ACK message, or a message with content (MSG).



Pathfinder

Pathfinder uses a recursive algorithm to check all possible paths from a given start point to a given end point using a given matrix. The program then returns the shortest path between these 2 points.

Data

Data exclusively passed to methods.

Methods

printMatrix()
prints the existing matrix

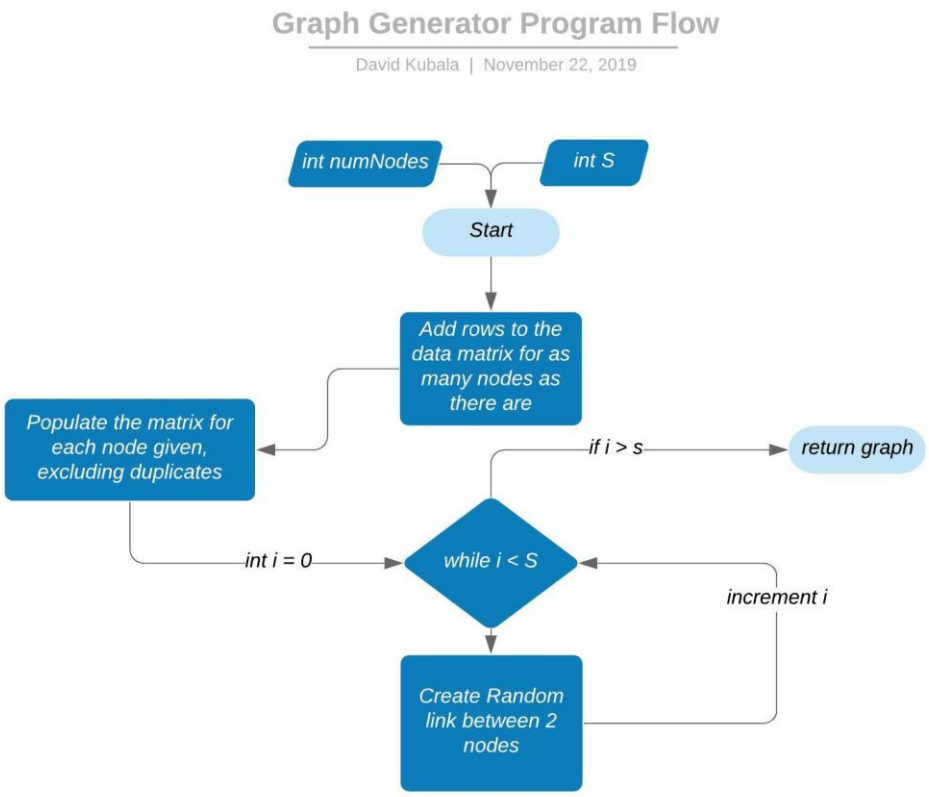
findPath()
returns the most efficient path from point a to b

findPathFast()
returns the first path the program finds, regardless of efficiency

The Pathfinder Class implements a recursive pathfinding algorithm, findPath(). The program takes in a given start and endpoint, and, using the graph described in the matrix, devises and returns the most efficient path by checking every possible path. The program then returns the shortest of these paths.

The program first takes in a matrix, an empty path, a start and end point (each as indices of the router represented) and checks if the start node is equal to the end node.

If the node is unequal to the end node, then it adds the node to the current path and scans for each nonzero entry in the given matrix row. If a nonzero entry is found and the entry has not already been visited, the method goes another layer deep on recursion until all possible paths have been explored.



GraphGenerator

GraphGenerator generates a random graph which is treated as a network over the course of the program.

Data

ArrayList<ArrayList<Integer>> *graph*

Methods

generateGraph()
generates a random graph

generateRandom()
generates a random point based on excluded values

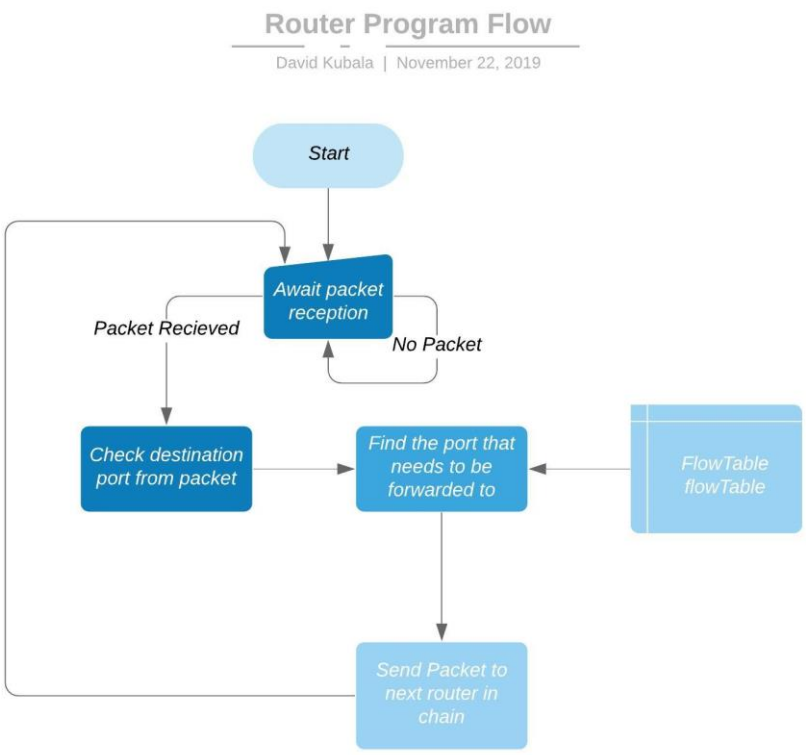
generateNames()
generates the names of each router in the network

The GraphGenerator Class takes in two operands: numNodes, which tells the graph how many routers should be in the network, and S, how many random connections should be added to the network.

The GraphGenerator first randomly creates and attaches nodes (in a virtual sense). This creates one long chain of nodes, which guarantees that the graph will be connected.

Next, the GraphGenerator links a number of S nodes at random. Once the random nodes are linked together, the program returns the fully filled in matrix.

In the final program, the efficiency of the Pathfinder is heavily dependent on the S variable; higher numbers of connections create exponentially more branches for the program to check.



Router

Router is the class that governs each router in the network. It includes a flow table, which is populated after it is created in the Controller class.

Data

FlowTable *flowTable*
InetSocketAddress *address*
int *port*
String *routerName*
Int *controllerPort*

Methods

onReciept()
thread constantly running on the router

findForwardPort()
finds the next port to forward data to based on the flow table and the destination of a packet

updateTable()
updates the flow table given a value

printFlowTable()
prints the Flow Table to the console

The Router class is mainly governed by the onReciept() method, which is a method that is constantly running on a separate thread while the Controller is running.

When the router wakes, it sends an opening message to the Controller, which is printed to the Controller terminal. The controller then sends an acknowledgement to the Router, ensuring its connection.

The Controller then creates the flow tables using the recursive pathfinding algorithm, and updates each router's flow tables accordingly.

When the router receives a packet from any source, it checks what the destination port of the packet is. Based on its flow table, it forwards the packet to the next router in the network towards the given destination.