# CPE 400: Computer Communication Networks

## Dynamic Routing Mechanism Design with Focus on Throughput

### Fall 2020, S. Sengupta

Created by Nick Alarez and Jayam Sutariya

Contents

[CPE 400: Computer Communication Networks 1](#_Toc58236053)

[Dynamic Routing Mechanism Design with Focus on Throughput 1](#_Toc58236054)

[Fall 2020, S. Sengupta 1](#_Toc58236055)

[Overview of Project Components 2](#_Toc58236056)

[Class: Router 2](#_Toc58236057)

[Private Member Variables 2](#_Toc58236058)

[Public Member Functions 2](#_Toc58236059)

[Public Member Variables 2](#_Toc58236060)

[Driver File 2](#_Toc58236061)

[Global Variables 2](#_Toc58236062)

[Initial Variables 2](#_Toc58236063)

[Functions 3](#_Toc58236064)

[main() Code 4](#_Toc58236065)

[Secondary Variables 5](#_Toc58236066)

Original README.md can be found at <https://github.com/nicky189/cpe400/>. Converted using Pandoc.

# Overview of Project Components

## Class: Router

### Private Member Variables

int ID //The router identifier  
double delayProcessing //Nodal processing delay (seconds)  
double delayTransmission //Link delay (seconds)  
double delayQueueing //Queueing delay (seconds)  
double delayPropagation //Propagation delay (seconds)  
double speedPropagation //Propagation speed of medium - set to 1 for simplicity of time reporting  
double lossProbability //Chance of losing packet en route to destination  
double bandwidth //Uplink speed, used in calculating propagation delay

### Public Member Functions

Router() //Default constructor  
Router(int id, int bSize, double d\_proc, double d\_trans, double s\_prop, double loss, double band) //Parameterized constructor  
~Router(); //Destructor  
void newLink(Router \* newRouter, int length) //Creates new connection on graph, taking in a Router object and distance  
double internalDelay() //Returns processing and queueing delay  
double timeOfTravel(Router \* dest, int packetSize) //Used to calculate packet transmission time by adding propagation delay and transmission delay. Propagation delay is calculated by dividing the size of packet by bandwidth and adding that to the length divided by propagation speed.  
int getID() //Returns router identifier

### Public Member Variables

vector<pair<Router\*, int> > routerLinks //Vector representation of the nodes the current Router can reach

## Driver File

### Global Variables

int destination //Destination router ID  
vector<Router\*> networkMesh //Stores graph representation  
vector<pair<double, int> > packetInfo //Tracks travel time and loss for each packet sent

### Initial Variables

Only variables that are not already listed under the Router class will be mentioned here.

int packetSize //Size of packet (bytes, default 256)  
double dDist //Stores distance between routers, only used as temp variable  
int lostPackets //Number of lost packets  
int main\_numberPackets //Packets to send (default 1)  
int origin //Source router ID  
int numberRouters //Number of routers in network (default 16)  
vector<vector<pair<int, int> > > linkDistances //Stores distances between routers  
char input //Stores user response on changing packet amount  
char v //Stores user response on changing verbose mode

### Functions

#### shortestPath()

vector<pair<int, int> > shortestPath(int startID, int dest, vector<vector<pair<int, int> > > routerLinks)

Creates a finalRoute set which contains the step number and the ID of the router. Creates a minDistance vector pair with distance (set to max, as Djikstra’s sets all unknown nodes to infinity) and router ID. Creates a delays vector pair with internal delay (likewise set to max) and router ID. The pair corresponding to the first router’s ID in minDistance is initialized to a distance of zero, since it is the beginning. Similarly, the same is done for delays except instead of zero, it is the internal delay. The finalRoute vector has a new node inserted (at the second position, technically) with the distance of zero and the source router ID.

A while loop runs as long as the finalRoute vector is not empty. A location variable is created corresponding to the startID. If the location of the packet is the same as the destination, the function exits and returns the minDistance vector which only contains the startID and a distance of zero. Otherwise, the finalRoute vector’s first element is erased, thus it becomes the element we inserted earlier.

A for loop is then entered (still within the while loop) which actually searches for the next path to take. This for loop is based off of routerLinks at location but is passed in as linkDistances, explained further below.

if ((minDistance[z.first].first > minDistance[location].first + z.second) || (delays[z.first].first > delays[location].first + networkMesh[z.first]->internalDelay()))

If the distance of the current route is greater than the minimum distance from the current router plus the distance to the first router that can be reached, OR the delay of the current route is greater than the delay of the current router plus the delay of the first router that can be reached, then…

(Note: This will always run at least once due to the length in each new minimumDistance link being set to INT\_MAX.)

finalRoute.erase({minDistance[z.first].first, z.first})

Whatever was in finalRoute at that point, erase it.

minDistance[z.first].first = minDistance[location].first + z.second

Update the minimum distance to our better path.

minDistance[z.first].second = location

Change the ID of the router that was there to our current router.

delays[z.first].first = delays[location].first + networkMesh[z.first]->internalDelay();  
 delays[z.first].second = location;

Liekwise, update the delay “path” and the ID of the router.

finalRoute.insert({minDistance[z.first].first, z.first})

Update our finalRoute with the new minimum distance.

After both loops complete, we return the minDistance vector.

#### printPath()

void printPath(vector<int> nodePath)

This function uses the nodePath vector to step through the packet’s route and print each iteration if verbose mode is enabled. Otherwise, a table printout of the time packets took to travel plus any lost packets is printed.

### main() Code

Initial variables are created and RNG is seeded. User is asked for source and destination router IDs and if they want to change the number of packets and enable verbose mode.

A for loop is entered corresponding to the number of packets.

The Router objects are created in a for loop using the initialized variables and each router is put into the networkMesh vector correlating to their IDs.

The network mesh is created. The graph is arbitrary.

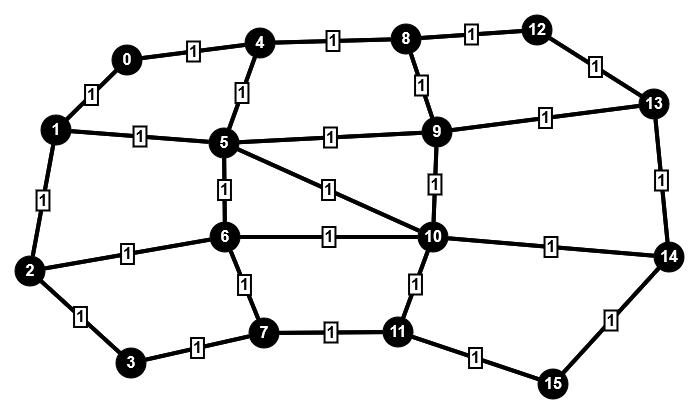


Image of Network Graph

A for loop runs corresponding to the number of routers. Within, the the totalLinks variable is created which takes on the value of the number of connections for a specific router. Another for loop runs for total number of links of that router and updates the dDist variable with the distance of connection. This is then put in the linkDistances vector alongside the ID of the router.

The shortestPath function runs and outputs its data into the pathInfo vector. To prepare for printing the path travelled, a prevRouter variable is initialized that holds the ID of the destination router. A nodePath vector is created. The destination router ID is pushed in followed by the router before it in the final path.

A while loop runs until it reaches the beginning of the path. The next prevRouter variable is set to the previous router of the previous router. This is then pushed back on the nodePath vector, so that when searched from the beginning, it starts at the source and ends at the destination.

### Secondary Variables

Router \* parent; //Used in calculating travel time  
Router \* child; //Used in calculating travel time  
bool lost = 0; //Lost packet flag  
double timer; //Running count of packet travel time  
int randMax = 100; //Max random value  
int randProb; //Random probability (out of 100)  
int lostPackets = 0 // Number of lost packets  
double timeFinal = 0 //Time of travel  
int droppedRouter = 0; //ID of router that caused dropped packet

A for loop runs corresponding to the size of nodePath, less 2, and the iterator x is decremented until we reach the first element.

The random probability is generated, and if that’s less than the chance of packetLoss, the program decides to drop a packet. If there is only one packet, the selected router is the current value of x. Otherwise, it is a random router from the nodePath. The lost flag is set to true.

Next, time of travel must be calculated. The parent and child Router objects are initialized to the current router and the one before it (that is, closer to source), respectively. The timer value is set to the result of the parent object calling timeOfTravel(). As the for loop executes, the timeFinal variable is updated each time.

Lastly, if there has been a lost packet, the for loop takes one step back to the previous router it ran for, increments lostPackets and resets the lost flag to false. Or, if there is more than one packet left, step the for loop back as well but decremement the number of packets left.

The for loop exits and the packetInfo vector is updated with the travel information. The route, travel time and number of lost packets are outputted if verbose mode is enabled.

All Router objects are destroyed so they receive new random values at next instantiation. The networkMesh and nodePath vectors are also cleared.

The for loop exits and the path is printed if verbose mode is disabled.