Project 1:

Dynamic Routing Mechanism with Focus on Throughput

By Joey Isola

Table of Contents

Objective

Create and simulate a new routing strategy that maximizes the overall throughput of a mesh network. Throughput is affected by many factors that should be considered, such as nodal processing delay, overloaded buffers, loss, etc. The more realistic assumptions you can make for your network, better it is.

Introduction

To create a network routing strategy that optimizes throughput, two important factors must be addressed: maximizing bandwidth and minimizing the number of jumps. This is accomplished by first building an undirected graph of interconnected nodes with weighted edges to simulate bandwidth measured in bits per second. Once the graph is generated, a depth first search is used to find all possible, non-looping paths from a given source node to destination node. Next using an adjacency table, the bandwidth of each path is calculated, and the maximum bandwidth of all available paths is chosen. If two paths have the same bandwidth, the path with the lower depth is kept helping minimize delay.

Protocol Functionality

A network of routers in constant communication must determine the best possible path from one to another in order to maximize throughput. Each router in this network is connected to adjacent routers with a predetermined bandwidth set for each connection. If router A wants to communicate with router B, it must find a path that avoids connections with lower bandwidth and minimize the number of jumps to help lessen delay.

The difference between bandwidth and throughput is that bandwidth is the potential of a connection’s speed where throughput is a real measurement of bits per second. The bandwidth of a path of connections is made up of the smallest edge bandwidth of the given path. Since a single edge of low bandwidth can ruin an entire path consisting of high bandwidth, more emphasis is put on avoiding slower edge bandwidths than finding the highest edge bandwidths. The path with the highest minimum edge bandwidth is chosen to reduce transmission delay which consists of packet size divided by rate of the link.

After a path's bandwidth is maximized, depth of the path is considered to reduce queueing and processing delay. In this simulation it is assumed that all routers share the same queuing and processing delay, so depth must be minimized after path bandwidth is maximized. Figure 1 shows the graph used to test the algorithm, where each edge weight is bandwidth, and each numbered node represents a router.

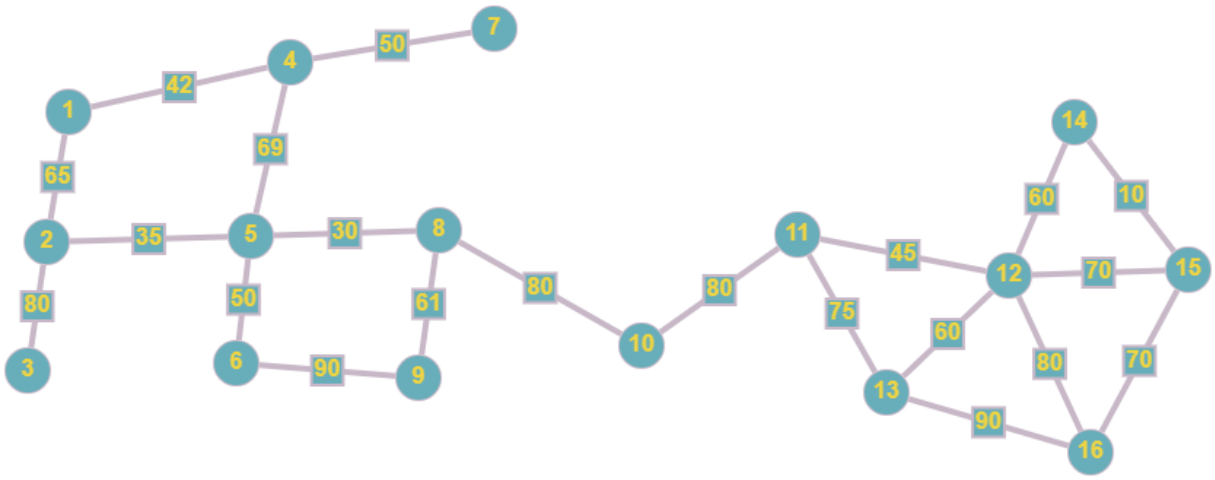

Figure 1: An undirected Graph of 16 Routers with bandwidth (bps) as weighted edges

Use Cases

To generate the network, a file named "data.txt" is read containing three values on each line representing in order: router A, router B, and edge bandwidth, as displayed by Figure 2.

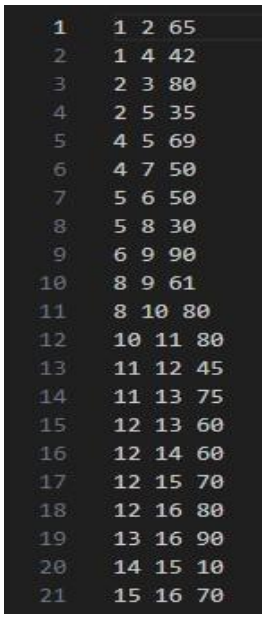

Figure 2: data.txt. Each entry follows the format: routerA routerB bandwidth.

The program will generate the graph with the information above using the "connectEdge(int, int, int)" function, which dynamically keeps track of vertices as they are added and fills an adjacency matrix with edge weights represented as bandwidths. After the data is read, the program will direct the user to input. Figure 3 displays options for user input.

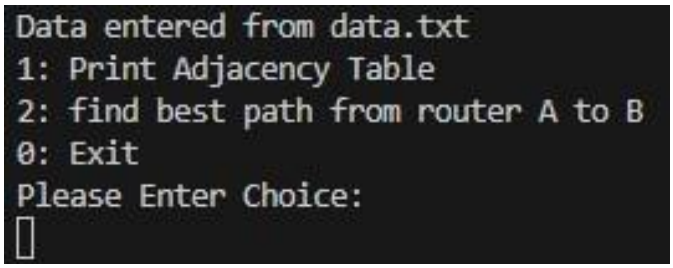

Figure 3: Input Options

With the data entered from Figure2, the adjacency table is shown in Table 1 after option "1" is selected.

Table 1: Adjacency Table

```
Please Enter Choice:
1
    |  1 |  2 |  3 |  4 |  5 |  6 |  7 |  8 |  9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
 1  |  0 | 65 |  0 | 42 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |
 2  | 65 |  0 | 80 |  0 | 35 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |
 3  |  0 | 80 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |
 4  | 42 |  0 |  0 |  0 | 69 |  0 | 50 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |
 5  |  0 | 35 |  0 | 69 |  0 | 50 |  0 | 30 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |
 6  |  0 |  0 |  0 |  0 | 50 |  0 |  0 |  0 | 90 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |
 7  |  0 |  0 |  0 | 50 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |
 8  |  0 |  0 |  0 |  0 | 30 |  0 |  0 |  0 | 61 | 80 |  0 |  0 |  0 |  0 |  0 |  0 |
 9  |  0 |  0 |  0 |  0 |  0 | 90 |  0 | 61 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |
10  |  0 |  0 |  0 |  0 |  0 |  0 |  0 | 80 |  0 |  0 | 80 |  0 |  0 |  0 |  0 |  0 |
11  |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 | 80 |  0 | 45 | 75 |  0 |  0 |  0 |
12  |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 | 45 |  0 | 60 | 60 | 70 | 80 |
13  |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 | 75 | 60 |  0 |  0 |  0 | 90 |
14  |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 | 60 |  0 |  0 | 10 |  0 |
15  |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 | 70 |  0 | 10 |  0 | 70 |
16  |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 | 80 | 90 |  0 | 70 |  0 |
```

After entering “2” on the options menu, the program prompts the user to enter source and destination nodes. Figure 4 shows every possible path from Node 5 to Node 12, then picks the path with the highest bandwidth with the shortest depth.

```
Please Enter Choice:
2
Please Enter Source Node: 5
Please Enter Destination Node: 12
Path 1: 5 6 9 8 10 11 12  Bandwidth: 45 Depth: 7
Path 2: 5 6 9 8 10 11 13 12  Bandwidth: 50 Depth: 8
Path 3: 5 6 9 8 10 11 13 16 12  Bandwidth: 50 Depth: 9
Path 4: 5 6 9 8 10 11 13 16 15 12  Bandwidth: 50 Depth: 10
Path 5: 5 6 9 8 10 11 13 16 15 14 12  Bandwidth: 10 Depth: 11
Path 6: 5 8 10 11 12  Bandwidth: 30 Depth: 5
Path 7: 5 8 10 11 13 12  Bandwidth: 30 Depth: 6
Path 8: 5 8 10 11 13 16 12  Bandwidth: 30 Depth: 7
Path 9: 5 8 10 11 13 16 15 12  Bandwidth: 30 Depth: 8
Path 10: 5 8 10 11 13 16 15 14 12  Bandwidth: 10 Depth: 9

Path with Highest Throughput: 5 6 9 8 10 11 13 12 Bandwidth: 50 Depth: 8
```

Figure 4: All Paths from Node 5 to Node 12 are shown along with bandwidth and depth, including the most efficient path

Error Handling

The most important error in this program is when the graph does not have at least one valid path for every node A and B in the graph. The algorithm assumes the graph is attached before processing data and will throw a segmentation fault if the graph is not constructed properly. Additionally, the number of vertices is set

dynamically as routers are added to the network, so all routers are expected to be added in sequentially.

The maximum number of routers on this network is set statically to 20, so any additional routers added will cause segmentation faults. If a source or destination is greater than 20 or less than 20 the program will throw an error. If the source is equal to the destination, the program will throw an error since depth first search in this context cannot find paths to itself. The terminal will only accept input as integers.

Novel Algorithms

The main algorithm in this program is FindAllPaths() which is a type of recursive depth first search that uses a helper function called pathsUtility(). The helper function recurses through every adjacent vertex that has not been visited on the current path and checks if the vertex is the destination node. The time complexity for this depth first search is $O(n^2)$. All paths that reach the destination node are stored in a 2-dimensional integer vector.

```
void Graph::pathsUtility(int aNode, int bNode, vector<int>& path){

    visited[aNode] = 1;
    path.push_back(aNode);

    if (aNode == bNode) allPaths.push_back(path);
    else {
        for(int i=1; i<=numVertices; i++){
            if((adjMatrix[aNode][i] != 0) && (visited[i] == 0)){
                pathsUtility(i, bNode, path);
            }
        }
    }
    path.pop_back();
    visited[aNode] = 0;


}

void Graph::findAllPaths(int sourceNode, int destNode){
    vector<int> path;

    pathsUtility(sourceNode, destNode, path);
}
```

Figure 5: Recursive Depth First Search Algorithm

Now that all valid paths are stored in a vector, the program must prioritize path bandwidth first, then depth of each path second. A structure called Info is created which holds 3 integers: id, depth, and bandwidth. Each valid path is organized into a vector of info objects via the setInfo() function, which iterates through each valid path and takes the minimum of each edge bandwidth as the path bandwidth. Each path's bandwidth and depth are stored. setInfo() has a time complexity of $O(n^2)$.

```
void Graph::setInfo(){
    int j = 0;
    for(const auto& path: allPaths){
        info tempInfo;
        int tempMin = 9999;
        tempInfo.depth = path.size();
        // cout << path.size() << endl;
        int num = path.size();
        // cout << path.size() << endl;
        for(int i = 0; i< num - 1; i++ ){

            tempMin = min(tempMin, adjMatrix[path[i]][path[i+1]]);
        }

        if(tempMin > maxBandwidth) maxBandwidth = tempMin;
        tempInfo.bandWidth = tempMin;
        tempInfo.id = j;
        j++;
        pathInfo.push_back(tempInfo);

    }
}
```

Figure 6: Algorithm to organize valid paths by bandwidth and depth

Next a function called findBestPath() iterates through the info object and selects the path with the lowest depth out of the paths with the highest bandwidth. FindBestPath() has a time complexity of O(n).

Results and Analysis

Using the data from Figure 1 and 2, the worst case this algorithm uses is finding paths from node 7 to node 15, which generates 40 paths. Figure 7 displays the pathfinding from node 7 to 15.

```
Please Enter Source Node: 7
Please Enter Destination Node: 15
Path 1: 7 4 1 2 5 6 9 8 10 11 12 13 16 15  Bandwidth: 35 Depth: 14
Path 2: 7 4 1 2 5 6 9 8 10 11 12 14 15  Bandwidth: 10 Depth: 13
Path 3: 7 4 1 2 5 6 9 8 10 11 12 15  Bandwidth: 35 Depth: 12
Path 4: 7 4 1 2 5 6 9 8 10 11 12 16 15  Bandwidth: 35 Depth: 13
Path 5: 7 4 1 2 5 6 9 8 10 11 13 12 14 15  Bandwidth: 10 Depth: 14
Path 6: 7 4 1 2 5 6 9 8 10 11 13 12 15  Bandwidth: 35 Depth: 13
Path 7: 7 4 1 2 5 6 9 8 10 11 13 12 16 15  Bandwidth: 35 Depth: 14
Path 8: 7 4 1 2 5 6 9 8 10 11 13 16 12 14 15  Bandwidth: 10 Depth: 15
Path 9: 7 4 1 2 5 6 9 8 10 11 13 16 12 15  Bandwidth: 35 Depth: 14
Path 10: 7 4 1 2 5 6 9 8 10 11 13 16 15  Bandwidth: 35 Depth: 13
Path 11: 7 4 1 2 5 8 10 11 12 13 16 15  Bandwidth: 30 Depth: 12
Path 12: 7 4 1 2 5 8 10 11 12 14 15  Bandwidth: 10 Depth: 11
Path 13: 7 4 1 2 5 8 10 11 12 15  Bandwidth: 30 Depth: 10
Path 14: 7 4 1 2 5 8 10 11 12 16 15  Bandwidth: 30 Depth: 11
Path 15: 7 4 1 2 5 8 10 11 13 12 14 15  Bandwidth: 10 Depth: 12
Path 16: 7 4 1 2 5 8 10 11 13 12 15  Bandwidth: 30 Depth: 11
Path 17: 7 4 1 2 5 8 10 11 13 12 16 15  Bandwidth: 30 Depth: 12
Path 18: 7 4 1 2 5 8 10 11 13 16 12 14 15  Bandwidth: 10 Depth: 13
Path 19: 7 4 1 2 5 8 10 11 13 16 12 15  Bandwidth: 30 Depth: 12
Path 20: 7 4 1 2 5 8 10 11 13 16 15  Bandwidth: 30 Depth: 11
Path 21: 7 4 5 6 9 8 10 11 12 13 16 15  Bandwidth: 45 Depth: 12
Path 22: 7 4 5 6 9 8 10 11 12 14 15  Bandwidth: 10 Depth: 11
Path 23: 7 4 5 6 9 8 10 11 12 15  Bandwidth: 45 Depth: 10
Path 24: 7 4 5 6 9 8 10 11 12 16 15  Bandwidth: 45 Depth: 11
Path 25: 7 4 5 6 9 8 10 11 13 12 14 15  Bandwidth: 10 Depth: 12
Path 26: 7 4 5 6 9 8 10 11 13 12 15  Bandwidth: 50 Depth: 11
Path 27: 7 4 5 6 9 8 10 11 13 12 16 15  Bandwidth: 50 Depth: 12
Path 28: 7 4 5 6 9 8 10 11 13 16 12 14 15  Bandwidth: 10 Depth: 13
Path 29: 7 4 5 6 9 8 10 11 13 16 12 15  Bandwidth: 50 Depth: 12
Path 30: 7 4 5 6 9 8 10 11 13 16 15  Bandwidth: 50 Depth: 11
Path 31: 7 4 5 8 10 11 12 13 16 15  Bandwidth: 30 Depth: 10
Path 32: 7 4 5 8 10 11 12 14 15  Bandwidth: 10 Depth: 9
Path 33: 7 4 5 8 10 11 12 15  Bandwidth: 30 Depth: 8
Path 34: 7 4 5 8 10 11 12 16 15  Bandwidth: 30 Depth: 9
Path 35: 7 4 5 8 10 11 13 12 14 15  Bandwidth: 10 Depth: 10
Path 36: 7 4 5 8 10 11 13 12 15  Bandwidth: 30 Depth: 9
Path 37: 7 4 5 8 10 11 13 12 16 15  Bandwidth: 30 Depth: 10
Path 38: 7 4 5 8 10 11 13 16 12 14 15  Bandwidth: 10 Depth: 11
Path 39: 7 4 5 8 10 11 13 16 12 15  Bandwidth: 30 Depth: 10
Path 40: 7 4 5 8 10 11 13 16 15  Bandwidth: 30 Depth: 9

Path with Highest Throughput: 7 4 5 6 9 8 10 11 13 12 15 Bandwidth: 50 Depth: 11
```

Figure 7: An analysis of the Algorithm's worst-case scenario

Since this algorithm is set up to only optimize pathfinding in networks with twenty or less routers, the exponential time complexity poses a problem to networks with a lot of routers. The algorithm excels in situations with a small number of routers, provided all routers added are connected.

The pathfinding algorithm successfully identifies the paths with the highest bandwidth and prioritizes them by depth. In this simulation, all routers are assumed to have an equal amount of queueing and processing delay, which makes depth an effective metric. Assuming that packet size is constant in every test, bandwidth is

the most important variable in determining transmission delay. This greedy strategy of assuming transmission delay is a higher priority than queueing and processing delay is most effective in situations where a network has low congestion, but a high variability in transmission delay.