COMPUTER NETWORKS
PROJECT

UNDERSTANDING NETWORK ROUTING PROBLEM AND STUDY OF ROUTING ALGORITHMS AND HEURISTICSTHROUGH IMPLEMENTATION

RHEEYA UPPAAL
3rd Year, Computer Science

Table of Contents

1. Introduction

1.1 Problem Statement

To identify, understand and compare various routing algorithms used in real world networks.

1.2. Objectives of Research

1. Define and understand the concepts of routing.

2. Determine if a Greedy or Dynammic Programming strategy algorithm is more efficient for routing, in general. Identify which strategy is used more in real world networks.

3. Identify the common routing algorithms used in networks. Identify which algorithms are used in which scenarios.

4. Identify the performance metrics for gauging algorithms.

5. Compare existing routing algorithms in various scenarios (on the simulation software). Also note specific phenomena or anomalies during simulation.

6. Think of modifications (if any) in existing routing algorithms, or devise a new routing algorithm.

2. Understanding the Routing Problem

2.1 What is routing?

The transport layer provides communication service between two processes running on two different hosts. In order to provide this service, the transport layer relies on the services of the network layer, which provides a communication service between hosts. In particular, the network-layer moves transport-layer segments from one host to another. At the sending host, the transport layer segment is passed to the network layer. In order to this, the network layer requires the coordination of each and every host and router in the network. In simple terms, if we have to define Routing in a lay man's language we can simply say that Routing is the manner/order in which we decide the path a segment shall follow from the sending host to the receiving one. This path includes a connection of links and routers. In technical terms though routing is a complex yet challenging concept.

Technically, Routing broadly consists of the following 3 functions:

1. Path Determination: This function determines the path/route the packets will follow from the sender to receiver. It involves various routing algorithms which are discussed further.

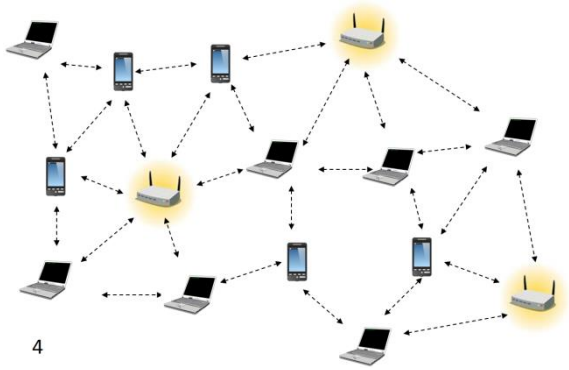
2. Switching: When a packet arrives at a router it needs to be further dispatched to other routers i.e. it is further switched to other routers.

3. Call Setup: Just like a TCP carries out 3 -way handshake similarly some network layer architectures (e.g., ATM) requires that the routers along the chosen path from source to destination handshake with each other in order to setup state before data actually begins to flow. In the network layer, this process is referred to as call setup.

The main goals of routing are:

1. Correctness: The routing should be done properly and correctly so that the packets may reach their proper destination.
2. Simplicity: The routing should be done in a simple manner so that the overhead is as low as possible. With increasing complexity of the routing algorithms the overhead also increases.
3. Robustness: Once a major network becomes operative, it may be expected to run continuously for years without any failures. The algorithms designed for routing should be robust enough to handle hardware and software failures and should be able to cope with changes in the topology and traffic without requiring all jobs in all hosts to be aborted and the network rebooted every time some router goes down.
4. Stability: The routing algorithms should be stable under all possible conditions.
5. Fairness: Every node connected to the network should get a fair chance of transmitting their packets. This is generally done on a first come first serve basis.

6. Optimality: The routing algorithms should be optimal in terms of throughput and minimizing mean packet delays. Here there is a trade-off and one has to choose depending on his suitability.



Routing is performed for many kinds of networks, including the telephone network (circuit switching), electronic data networks (such as theInternet), and transportation networks. This article is concerned primarily with routing in electronic data networks using packet switching technology.

In packet switching networks, routing directs packet forwarding (the transit of logically addressed network packets from their source toward their ultimate destination) through intermediate nodes. Intermediate nodes are typically network hardware devices such as routers, bridges, gateways, firewalls, or switches. General-purpose computers can also forward packets and perform routing, though they are not specialized hardware and may suffer from limited performance. The routing process usually directs forwarding on the basis of routing tables, which maintain a record of the routes to various network destinations. Thus, constructing routing tables, which are held in the router's memory, is very important for efficient routing. Most routing algorithms use only one network path at a time. Multipath routing techniques enable the use of multiple alternative paths.

In case of overlapping/equal routes, algorithms consider the following elements to decide which routes to install into the routing table (sorted by priority):

1. Prefix-Length: where longer subnet masks are preferred (independent of whether it is within a routing protocol or over different routing protocol)
2. Metric: where a lower metric/cost is preferred (only valid within one and the same routing protocol)
3. Administrative distance: where a route learned from a more reliable routing protocol is preferred (only valid between different routing protocols)

2.2 Routing Algorithms

2.2.1 Algorithmic Strategies used for Routing

- Brute force algorithm

- Greedy strategy
- Dynamic programming
- Backtracking
- Branch and Bound
- Divide and Conquer
- Decrease and Conquer
- Transfer and Conquer

2.2.2 Types of Routing Algorithms

1. Link state routing:

Link-state routing protocols are one of the two main classes of routing protocols used in packet switching networks for computer communications, the other being distance-vector routing protocols. Examples of link-state routing protocols include open shortest path first (OSPF) and intermediate system to intermediate system (IS-IS).The link-state protocol is performed by every switching node in the network (i.e., nodes that are prepared to forward packets; in the Internet, these are called routers).

The basic concept of link-state routing is that every node constructs a map of the connectivity to the network, in the form of a graph, showing which nodes are connected to which other nodes. Each node then independently calculates the next best logical path from it to every possible destination in the network. The collection of best paths will then form the node's routing table.This contrasts with distance-vector routing protocols, which work by having each node share its routing table with its neighbours. In a link-state protocol the only information passed between nodes is connectivity related.

Strategy used : greedy programming, generally a variant of Dijkstra's algorithm is used.

2. Distance vector routing:

In computer communication theory relating to packet-switched networks, a distance-vector routing protocol is one of the two major classes of intra domain routing protocols, the other major class being the link-state protocol.

Distance-vector routing protocols use the Bellman–Ford algorithm, Ford–Fulkerson algorithm, or DUAL FSM (in the case of Cisco Systems's protocols) to calculate paths. A distance-vector routing protocol requires that a router inform its neighbors of topology changes periodically. Compared to link-state protocols, which require a router to inform all the nodes in a network of topology changes, distance-vector routing protocols have less computational complexity and message overhead.

The term distance vector refers to the fact that the protocol manipulates vectors (arrays) of distances to other nodes in the network. The vector distance algorithm was the original ARPANET routing algorithm and was also used in the internet under the name of RIP (Routing Information Protocol).

Examples of distance-vector routing protocols include RIPv1 and IGRP.
Strategy used : dynamic programming, generally bellman ford algorithm.

2.2.1 Common Routing Algorithms

The shortest paths are calculated using suitable algorithms on the graph representations of the networks. Let the network be represented by graph G (V, E) and let the number of nodes be 'N'. For all the algorithms discussed below, the costs associated with the links are assumed to be positive. A node has zero cost w.r.t itself. Further, all the links are assumed to be symmetric, i.e. if $d_{i,j}$ = cost of link from node i to node j, then $d_{i,j} = d_{j,i}$. The graph is assumed to be complete. If there exists no edge between two nodes, then a link of infinite cost is assumed. The algorithms given below find costs of the paths from all nodes to a particular node; the problem is equivalent to finding the cost of paths from a source to all destinations.

1. Bellman-Ford Algorithm

This algorithm iterates on the number of edges in a path to obtain the shortest path. Since the number of hops possible is limited (cycles are implicitly not allowed), the algorithm terminates giving the shortest path.

Notation:

$d_{i,j}$ = Length of path between nodes i and j, indicating the cost of the link.
h = Number of hops.
D[i,h] = Shortest path length from node i to node 1, with upto 'h' hops.
D[1,h] = 0 for all h .

Algorithm :

Initial condition : D[i, 0] = infinity, for all i (i != 1)
Iteration : $D[i, h+1] = \min \{ d_{i,j} + D[j,h] \}$ over all values of j .
Termination : The algorithm terminates when
D[i, h] = D [i, h+1] for all i .

Principle:
For zero hops, the minimum length path has length of infinity, for every node. For one hop the shortest-path length associated with a node is equal to the length of the edge between that node and node 1. Hereafter, we increment the number of hops allowed, (from h to h+1) and find out whether a shorter path exists through each of the other nodes. If it exists, say through node 'j', then its length must be the sum of the lengths between these two nodes (i.e.

$d_{i,j}$) and the shortest path between j and 1 obtainable in upto h paths. If such a path doesn't exist, then the path length remains the same. The algorithm is guaranteed to terminate, since there are utmost N nodes, and so N-1 paths. It has time complexity of O (N^3) .

2. Dijkstra's Algorithm

Notation:
D_i = Length of shortest path from node 'i' to node 1.
$d_{i,j}$ = Length of path between nodes i and j .

Algorithm:
Each node j is labeled with Dj, which is an estimate of cost of path from node j to node 1. Initially, let the estimates be infinity, indicating that nothing is known about the paths. We now iterate on the length of paths, each time revising our estimate to lower values, as we obtain them. Actually, we divide the nodes into two groups ; the first one, called set P contains the nodes whose shortest distances have been found, and the other Q containing all the remaining nodes. Initially P contains only the node 1. At each step, we select the node that has minimum cost path to node 1. This node is transferred to set P. At the first step, this corresponds to shifting the node closest to 1 in P. Its minimum cost to node 1 is now known. At the next step, select the next closest node from set Q and update the labels corresponding to each node using :$D_j = \min [D_j , D_i + d_{j,i}]$. Finally, after N-1 iterations, the shortest paths for all nodes are known, and the algorithm terminates.

Principle
Let the closest node to 1 at some step be i. Then i is shifted to P. Now, for each node j , the closest path to 1 either passes through i or it doesn't. In the first case Dj remains the same. In the second case, the revised estimate of D_j is the sum $D_i + d_{i,j}$. So we take the minimum of these two cases and update D_j accordingly. As each of the nodes get transferred to set P, the estimates get closer to the lowest possible value. When a node is transferred, its shortest path length is known. So finally all the nodes are in P and the D_j's represent the minimum costs. The algorithm is guaranteed to terminate in N-1 iterations and its complexity is O(N^2).

3. The Floyd Warshall Algorithm

This algorithm iterates on the set of nodes that can be used as intermediate nodes on paths. This set grows from a single node (say node 1) at start to finally all the nodes of the graph. At each iteration, we find the shortest path using given set of nodes as intermediate nodes, so that finally all the shortest paths are obtained.

It is observed that all the three algorithms mentioned above give comparable performance, depending upon the exact topology of the network.

3. Comparison of Algorithms through Implementation

3.1 Performance Metrics for Comparison

Router metrics are metrics used by a router to make routing decisions. It is typically one of many fields in a routing table.

Metrics are used to determine whether one route should be chosen over another. The routing table stores possible routes, while link-state or topological databases may store all other information as well. For example, Routing Information Protocol uses hopcount (number of hops) to determine the best possible route. The route will go in the direction of the gateway with the lowest metric. The direction with the lowest metric can be a default gateway.

Router metrics can contain any number of values that help the router determine the best route among multiple routes to a destination. A router metric typically based on information like path length, bandwidth, load, hop count, path cost, delay, Maximum Transmission Unit (MTU), reliability and communications cost.

A Metric can include:

- measuring link utilization (using SNMP)
- number of hops (hop count)
- speed of the path
- packet loss (router congestion/conditions)
- latency (delay)
- path reliability
- path bandwidth
- throughput [SNMP - query routers]
- load
- MTU

The six evaluation criteria when determining the efficiency of an algorithm

1. THROUGHPUT: In general terms, throughput is the rate of production or the rate at which something can be processed. When used in the context of computer networking, such as Ethernet or packet radio, throughput or network throughput is the rate of successful message delivery over a communication channel. The data these messages belong to may be delivered over a physical or logical link or it can pass through a

certain network node/router. Throughput is usually measured in bits per second (bit/s or bps), and sometimes in data packets per second (p/s or pps).The system throughput or aggregate throughput is the sum of the data rates that are delivered to all terminals in a network. It can be analyzed mathematically by applying the queueing theory, where the load in packets per time unit is denoted as the arrival rate (λ), and the throughput,in packets per time unit, is denoted as the departure rate (μ).

2. GOODPUT: In computer networks, good put is the application level throughput, i.e. The number of useful information bits delivered by the network to a certain destination per unit of time. The amount of data considered excludes protocol overhead bits as well as retransmitted data packets. This is related to the amount of time from the first bit of the first packet sent (or delivered) until the last bit of the last packet is delivered. For example, if a file is transferred, the good put that the user experiences corresponds to the file size in bits divided by the file transfer time. The good put is always lower than the throughput (the gross bit rate that is transferred physically), which generally is lower than network access connection speed.

3. NETWORK LATENCY: Network latency in a packet-switched network is measured either one-way (the time from the source sending a packet to the destination receiving it), or round-trip delay time (the one-way latency from source to destination plus the one-way latency from the destination back to the source). It further consists of the processing delay, queuing delay and transmission delay. The processing delay is basically the time a sender host takes to process a packet and identify the router. Once the router is identified, queuing delay is encountered when a packet has to wait in the queuing buffer before it is transferred further. Transmission delay consists of the time to transmit the packet over the link.

4. LINK CAPACITY: The term link capacity defines the net bit rate (aka. Peak bit rate, information rate, or physical layer useful bit rate), or the maximum throughput of a logical or physical communication path in a digital communication system. For example, bandwidth tests measure the maximum throughput of a computer network.

5. NUMBER OF BOTTLENECKS: Bottleneck basically means traffic/congestion at various points in the network link. The number of bottlenecks signifies the number of place throughout the network link where a bottleneck has occurred.

6. TRAFFIC INTENSITY: In a digital network, the traffic intensity measures the ratio of the arrival rate of packets to the average packet length. Is: (al)/R where a is the average arrival rate of packets (e.g. In packets per second), L is the average packet length (e.g. In bits), and R is the transmission rate (e.g. Bits per second).

Performance metrics selected for the implementation of this project:

1. Throughput
2. Delay

3.2 Software and Testing Environment

In communication and computer network research, network simulation is a technique where a program models the behaviour of a network either by calculating the interaction between the different network entities (hosts/packets, etc.) using mathematical formulas, or actually capturing and playing back observations from a production network. The behaviour of the network and the various applications and services it supports can then be observed in a test lab; various attributes of the environment can also be modified in a controlled manner to assess how the network would behave under different conditions.

A network simulator is software that predicts the behaviour of a computer network. Since communication Networks have become too complex for traditional analytical methods to provide an accurate understanding of system behaviour network simulator are used. In simulators, the computer network is typically modelled with devices, links, applications etc. and the performance is analysed. Simulators typically come with support for the most popular technologies and networks in use today.

Most of the commercial simulators are GUI driven, while some network simulators are CLI driven. The network model/configuration describes the state of the network (nodes, routers, switches, links) and the events (data transmissions, packet error etc.). An important output of simulations are the trace files. Trace files log every packet, every event that occurred in the simulation and are used for analysis. Network simulators can also provide other tools to facilitate visual analysis of trends and potential trouble spots.

Simulation of networks is a very complex task. For example, if congestion is high, then estimation of the average occupancy is challenging because of high variance. To estimate the likelihood of a buffer overflow in a network, the time required for an accurate answer can be extremely large. Specialized techniques such as "control variates" and "importance sampling" have been developed to speed simulation.

The network simulator must enable a user to:

- Model the network topology specifying the nodes on the network and the links between those nodes
- Model the application flow (traffic) between the nodes
- Providing network performance metrics as output
- Visualization of the packet flow
- Logging of packet / events for drill down analyses / debugging

The “ns-3” simulation software is built using C++ and Python with scripting capability. The ns-3 library is wrapped by Python thanks to the pybindgen library which delegates the parsing of the ns-3 C++ headers to gccxml and pygccxml to automatically generate the corresponding C++ binding glue. These automatically-generated C++ files are finally

compiled into the ns-3 Python module to allow users to interact with the C++ ns-3 models and core through Python scripts. The ns-3 simulator features an integrated attribute-based system to manage default and per-instance values for simulation parameters. All of the configurable default values for parameters are managed by this system, integrated with command-line argument processing. The large majority of its users focuses on wireless simulations which involve models for Wi-Fi.

3.3 Network Characteristics and Topology

The general process of creating a simulation can be divided into several steps:

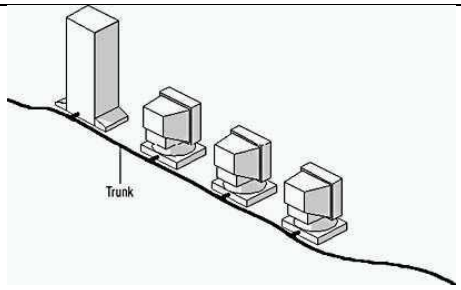
1. Topology definition: to ease the creation of basic facilities and define their interrelationships, ns-3 has a system of containers and helpers that facilitates this process.
2. Model development: models are added to simulation (for example, UDP, IPv4, point-to-point devices and links, applications); most of the time this is done using helpers.
3. Node and link configuration: models set their default values (for example, the size of packets sent by an application or MTU of a point-to-point link); most of the time this is done using the attribute system.
4. Execution: simulation facilities generate events, data requested by the user is logged.
5. Performance analysis: after the simulation is finished and data is available as a time-stamped event trace. This data can then be statistically analysed with tools like R to draw conclusions.
6. Graphical Visualization: raw or processed data collected in a simulation can be graphed using tools like Gnuplot, matplotlib or XGRAPH.

The selection of a network topology can affect:

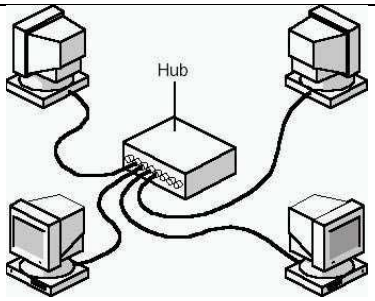
- Type of equipment the network needs.
- Capabilities of the equipment.
- Growth of the network.
- Way the network is managed.

Standard Topologies:

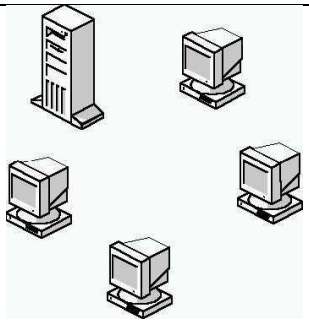
- Bus – Devices connected to a common, shared cable.
- Star - Connecting computers to cable segments branch out from a single point, or hub.
- Ring - Connecting computers to cable that form a loop.
- Mesh – Connects all computers in a network to each other with separate cables.



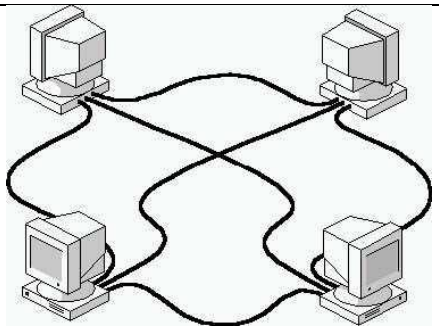
Bus topology network



Simple star network

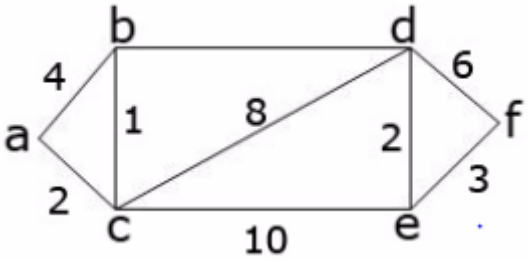


Simple ring network showing logical ring



Mesh topology

The topology of the network used during simulation is as follows:



w(u,v)	a	b	c	d	d	f
a	0	4	2	INF	INF	INF
b	4	0	1	5	INF	INF
c	2	1	0	8	10	INF
d	INF	5	8	0	2	6
e	INF	INF	10	2	0	3
f	INF	INF	INF	6	3	0

3.4 Routing Algorithms Used

Since we are finding the minimum path in the above 2 algorithms, therefore we consider delay as our cost in the above graphs because we want to minimize the delay in any routing algorithm. In general we consider that performance metric as our parameter which we want to minimize.
On the basis of minimum delay, the other performance metrics can be analysed further.

3.4.1 Dijkstra's Algorithm

Dijkstra's Algorithm for a single source shortest path is a:

- Link state algorithm
- Greedy algorithm
- Used to implement the OSPF (Open Shortest Path First) protocol

It thus neatly represents two categories of algorithms and a protocol, making a clear comparison in three different areas.

Advantages:

- Quickly adjusts to link failures
- Does not propagate the entire routing protocol but it transmits information only about its link
- Suitable for large networks
- Fast for fault discovery and rerouting.

Disadvantages:

- Complicated to configure
- Consumes large bandwidth
- Requires higher processing and memory

Code:

```
#include<iostream>
#define INFINITY 999
using namespace std;

classDijkstra{
private:
intadjMatrix[15][15];
int predecessor[15],distance[15];
bool mark[15]; //keep track of visited node
int source;
intnumOfVertices;
public:
```

```
void read();
/* Function read() reads No of vertices, Adjacency Matrix and source Matrix from the user.
The number of vertices must be greather than zero, all members of Adjacency Matrix must
be postive as distances are always positive. The source vertex must also be positive from 0to
noOfVertices - 1    */

void initialize();
/* Function initialize initializes all the data members at the begining of the execution. The
distance between source to source is zero and all other distances between source and
vertices are infinity. The mark is initialized to false and predecessor is initialized to -1   */

intgetClosestUnmarkedNode();
 /*Function getClosestUnmarkedNode returns the node which is nearest from the
Predecessor marked node. If the node is already marked as visited, then it search for
another node. */

voidcalculateDistance();
 /*Function calculateDistance calculates the minimum distances from the source node to
Other node. */

/*Function output prints the results */
void output();
voidprintPath(int);
};


voidDijkstra::read(){
cout<<"Enter the number of vertices of the graph(should be > 0)\n";
cin>>numOfVertices;
while(numOfVertices<= 0) {
cout<<"Enter the number of vertices of the graph(should be > 0)\n";
cin>>numOfVertices;
   }
cout<<"Enter the adjacency matrix for the graph\n";
cout<<"To enter infinity enter "<<INFINITY<<endl;
for(int i=0;i<numOfVertices;i++) {
cout<<"Enter the (+ve)weights for the row "<<i<<endl;
for(int j=0;j<numOfVertices;j++) {
cin>>adjMatrix[i][j];
while(adjMatrix[i][j]<0) {
cout<<"Weights should be +ve. Enter the weight again\n";
cin>>adjMatrix[i][j];
         }
      }
   }
cout<<"Enter the source vertex\n";
cin>>source;
```

```
while((source<0) && (source>numOfVertices-1)) {
cout<<"Source vertex should be between 0 and"<<numOfVertices-1<<endl;
cout<<"Enter the source vertex again\n";
cin>>source;
   }
}

voidDijkstra::initialize(){
for(int i=0;i<numOfVertices;i++) {
mark[i] = false;
predecessor[i] = -1;
distance[i] = INFINITY;
   }
distance[source]= 0;
}

intDijkstra::getClosestUnmarkedNode(){
intminDistance = INFINITY;
intclosestUnmarkedNode;
for(int i=0;i<numOfVertices;i++) {
if((!mark[i]) && ( minDistance>= distance[i])) {
minDistance = distance[i];
closestUnmarkedNode = i;
      }
   }
returnclosestUnmarkedNode;
}

voidDijkstra::calculateDistance(){
initialize();
intminDistance = INFINITY;
intclosestUnmarkedNode;
int count = 0;
while(count <numOfVertices) {
closestUnmarkedNode = getClosestUnmarkedNode();
mark[closestUnmarkedNode] = true;
for(int i=0;i<numOfVertices;i++) {
if((!mark[i]) && (adjMatrix[closestUnmarkedNode][i]>0) ) {
if(distance[i] > distance[closestUnmarkedNode]+adjMatrix[closestUnmarkedNode][i]) {
distance[i] = distance[closestUnmarkedNode]+adjMatrix[closestUnmarkedNode][i];
predecessor[i] = closestUnmarkedNode;
            }
         }
      }
count++;
   }
}
```

```
voidDijkstra::printPath(int node){
if(node == source)
cout<<(char)(node + 97)<<"..";
else if(predecessor[node] == -1)
cout<<"No path from “<<source<<”to "<<(char)(node + 97)<<endl;
else {
printPath(predecessor[node]);
cout<<(char) (node + 97)<<"..";
   }
}

voidDijkstra::output(){
for(int i=0;i<numOfVertices;i++) {
if(i == source)
cout<<(char)(source + 97)<<".."<<source;
else
printPath(i);
cout<<"->"<<distance[i]<<endl;
   }
}


int main(){
Dijkstra G;
G.read();
G.calculateDistance();
G.output();
return 0;
}
```

Output:

```
To enter infinity enter 999
Enter the (+ve)weights for the row 0
0 4 2 999 999 999
Enter the (+ve)weights for the row 1
4 0 1 5 999 999
Enter the (+ve)weights for the row 2
2 1 0 8 10 999
Enter the (+ve)weights for the row 3
999 5 8 0 2 6
Enter the (+ve)weights for the row 4
999 999 10 2 0 3
Enter the (+ve)weights for the row 5
999 999 999 6 3 0
Enter the source vertex
0
a..0->0
a..c..b..->3
a..c..->2
a..c..b..d..->8
a..c..b..d..e..->10
a..c..b..d..e..f..->13
```

3.4.2 Bellman Ford Algorithm

The Bellman-Ford Algorithm for a shortest path is a:

- Distance vector algorithm
- Dynamic programming algorithm
- Used to implement the RIP (Routing Information Protocol) protocol

It thus neatly represents two categories of algorithms and a protocol, making a clear comparison in three different areas.

Advantages:

- Simple
- Easy to configure

Disadvantages:

- Not scalable in heterogeneous networks
- The periodic updating of routing table consumes bandwidth because the entire routing table is propagated to neighbours
- Slow convergence
- Not suitable for large networks

Code:

```
//The topology could be changed or entered by the user
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>
// a structure to represent a weighted edge in graph
struct Edge
{      intsrc, dest, weight;};
// a structure to represent a connected, directed and weighted graph
struct Graph
{ // V-> Number of vertices, E-> Number of edges
int V, E;
  // graph is represented as an array of edges.
struct Edge* edge;
};
// Creates a graph with V vertices and E edges
struct Graph* createGraph(int V, int E)
{
struct Graph* graph = (struct Graph*) malloc(sizeof(struct Graph));
graph->V = V;
graph->E = E;
graph->edge = (struct Edge*) malloc(graph->E * sizeof(struct Edge));
```

```
return graph;
}
// A utility function used to print the solution
voidprintArr(intdist[], int n)
{
printf("Vertex   Distance from Source\n");
for (int i = 0; i < n; ++i)
printf("%d \t\t %d\n", i, dist[i]);
}
// The main function that finds shortest distances from src to all other
// vertices using Bellman-Ford algorithm.  The function also detects negative
// weight cycle
voidBellmanFord(struct Graph* graph, intsrc)
{
int V = graph->V;
int E = graph->E;
intdist[V];
  // Step 1: Initialize distances from src to all other vertices as INFINITE
for (int i = 0; i < V; i++)
dist[i] = INT_MAX;
dist[src] = 0;
 // Step 2: Relax all edges |V| - 1 times. A simple shortest path from src
    // to any other vertex can have at-most |V| - 1 edges
for (int i = 1; i <= V - 1; i++)
   {
for (int j = 0; j < E; j++)
       {
int u = graph->edge[j].src;
int v = graph->edge[j].dest;
int weight = graph->edge[j].weight;
if (dist[u] != INT_MAX &&dist[u] + weight <dist[v])
dist[v] = dist[u] + weight;
       }
   }
   // Step 3: check for negative-weight cycles.  The above step guarantees
   // shortest distances if graph doesn't contain negative weight cycle.
   // If we get a shorter path, then there is a cycle.
for (int i = 0; i < E; i++)
   {
int u = graph->edge[i].src;
int v = graph->edge[i].dest;
int weight = graph->edge[i].weight;
if (dist[u] != INT_MAX &&dist[u] + weight <dist[v])
printf("Graph contains negative weight cycle"); }
printArr(dist, V);
return;}
// Driver program to test above functions
```

```
int main()
{
int V = 5; // Number of vertices in graph
int E = 8; // Number of edges in graph
struct Graph* graph = createGraph(V, E);
    // add edge 0-1 (or A-B in above figure)
graph->edge[0].src = 0;
graph->edge[0].dest = 1;
graph->edge[0].weight = -1;
   // add edge 0-2 (or A-C in above figure)
graph->edge[1].src = 0;
graph->edge[1].dest = 2;
graph->edge[1].weight = 4;
  // add edge 1-2 (or B-C in above figure)
graph->edge[2].src = 1;
graph->edge[2].dest = 2;
graph->edge[2].weight = 3;
   // add edge 1-3 (or B-D in above figure)
graph->edge[3].src = 1;
graph->edge[3].dest = 3;
graph->edge[3].weight = 2;
    // add edge 1-4 (or A-E in above figure)
graph->edge[4].src = 1;
graph->edge[4].dest = 4;
graph->edge[4].weight = 2;
   // add edge 3-2 (or D-C in above figure)
graph->edge[5].src = 3;
graph->edge[5].dest = 2;
graph->edge[5].weight = 5;
    // add edge 3-1 (or D-B in above figure)
graph->edge[6].src = 3;
graph->edge[6].dest = 1;
graph->edge[6].weight = 1;
  // add edge 4-3 (or E-D in above figure)
graph->edge[7].src = 4;
graph->edge[7].dest = 3;
graph->edge[7].weight = -3;
BellmanFord(graph, 0);
return 0;
}
```

Output:

```
Vertex   Distance from Source
0                0
1                -1
2                2
3                -2
4                1
```

3.5 Observations

Experimental results (simulation) in the IPv4 network protocol uses RIPv2 and OSPFv2 by using two different simulators is GNS3. The result that the speed OSPFv2 router for inter-router converge better than RIPv2 routers in the experiment with GNS3.

In experiments with GNS3 time from R converge on IP 192.168.5.2 which uses OSPFv2 router, round-trip min / avg / max = 996/1142/1200 ms, and the process of tracing the route from R5 to R1 which is headed to the IP 192.168. 1.1 through 192.168.6.2 takes 1060 msec while through the IP 192.168.5.2 takes 340 msec and through IP 192.168.2.2 takes 1768 msec.

For RIPv2 routers show round-trip min / avg / max = 924/1292/1440 and processes tracing the route from R1 to R5 is heading to IP 192.168.1.1 through 192.168.6.2 takes 1460 msec while through the IP 192.168.5.2 takes 884 msec and through IP 192.168.2.2 takes 1972msec.

RIP multicast method takes a long time in terms of packet delivery.

3.6 Inferences

From the description and comparison of performance as well as the experimental results OSPFv2 Routing Protocol (OPEN Shortest Path First version 2) and RIPv2 (Routing Information Protocol version 2) in the IPv4 network, then it can be concluded that:

1. Every router within the same routing protocols build routing tables, based on information from neighboring routers for sharing information between routers.
2. Based on the speed of delivery of the package with the parameter used is the time between networks that converge OSPFv2 routing protocols rather than RIPv2 better use
3. RIPv2 using distance / hops while for OSPF willuse the same area thus saving bandwidth usage
4. ENSP Simulator GNS3 looks faster than the time required for inter-network konverge
5. To wider network then it would be better to use Dijkstra routers because of its ability to divide the network area into several sections.

4. Conclusions and Future Scope

We can conclude that Dijkstra's Algorithm outperforms the Bellmnan-Ford algorithm in terms of average throughput and instant packet delay in different sizes of network. In terms of number of packets lost, Dijkstra's algorithm is better compared to Bellaman-Ford algorithm in small networks but Bellaman-Ford is better in large networks.

OSPF is better than RIP for many reasons: OSPF uses either bandwidth or delay as metric for shortest path and it does not use the number of hops as in RIP.

OSPF can adjust the link and OSPF coverage network more quickly than RIP, but if RIP is enhanced by using FS-RIP, then RIP offers a better performance than OSPF. The work can be extended to other dynamic routing protocols and implemented by NS2. It can also be extended to evaluate other routing protocol criteria such as CPU utilization, jitter, and ability to provide Quality of Service (QoS).

This project provides further ground for other comparisons to be made in more diverse networks on different simulation software. The algorithms considered in this project can be studied and compared in greater detail.

5. References

- Kurose-Ross, “Computer Networking a TOP-Down Approach, Fifth edition, Pearson Education.
- Setiawati L. Differences Performance OSPFv2 and RIPv2 Routing Protocols In Network IPv4 Using Simulator GNS3 And ENSP
- Wu, B. (2011). Simulation Based Performance Analyses on RIPv2, EIGRP, and OSPF Using OPNET.
- Youssef, M., Younis, M. F., &Arisha, K. (2002, March). A constrained shortest-path energy-aware routing algorithm for wireless sensor networks. In *Wireless Communications and Networking Conference, 2002. WCNC2002. 2002 IEEE*(Vol. 2, pp. 794-799). IEEE.
- Liu, L., Jin, J., Palaniswami, M., Liu, M., Li, X., & Huang, Z. (2012). *Graph-Based Routing, Broadcasting and Organizing Algorithms for Ad-Hoc Networks*. INTECH Open Access Publisher.
- Boominathan, P., &Arora, K. Routing Planning As An Application Of Graph Theory.
- Lefkowitz, Corbis “Introduction to Graph Theory”
- Klampfer, S., Mohorko, J., Cucej, Z., &Chowdhury, A. (2012). Graph’s theory approach for searching the shortest routing path in RIP protocol: a case study.*PRZEGLAD ELEKTROTECHNICZNY, 88*(8), 224-231.
- Chiang, S. S., Huang, C. H., & Chang, K. C. A Minimum Hop Routing Protocol for Wireless Sensor Networks.
- Kassabalidis, I., Das, A. K., El-Sharkawi, M. A., Marks II, R. J., Arabshahi, P., & Gray, A. (2001, August). Intelligent routing and bandwidth allocation in wireless networks. In *Proc. NASA Earth Science Technology Conf. College Park, MD, August 28* (Vol. 30).
- Devi, G. S., Kumar, G. S., G D, P. V., & Reddy, P. (2011). Minimum Hop Energy Efficient Routing Protocol. *International Journal of Computer Applications*, *34*(4).
- Information shared with us by Dr. HimanshuAgarwal
- www.wikipedia.org
- www.stackoverflow.com